CS-GY 6083 A: Principles of Database Systems

Relational Algebra and SQL

supplementary material:
"Database Management Systems" Sec. 3.6, 4.1, 4.2, 5.1-5.6
class notes

code: Sailors.sql, Query.sql
Animals.sql, Query2.sql

Prof. Julia Stoyanovich
Computer Science and Engineering at Tandon
Center for Data Science
New York University

Game of Thrones

Characters

name	house
Eddard	Stark
Jon Snow	Stark
Tyrion	Lannister
Daenerys	Targaryen

Episodes

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

Appearances

name	season	num
Eddard	1	1
Eddard	1	2
Jon Snow	1	2
Jon Snow	2	1
Jon Snow	2	2
Tyrion	1	1
Tyrion	1	2
Tyrion	2	2
Daenerys	1	2
Daenerys	2	1

```
select E.title, C.name, C.house
from   Characters C, Episodes E, Appearances A
where  C.house = 'Targaryen'
and    A.name = C.name
and    E.season = A.season
and    E.num = A.num
```

What does this query compute?
How does it compute its results?

Outline

- Part 1: Relational algebra
- Part 2: SQL (structured query language)
  - Operations on a single relation
  - Combining relations
  - Aggregation
  - Additional SQL features

Select-project-join SQL queries

required

optional

```
select <attribute list>
from   <relation list>
where  <condition>
and    <condition>
and    ...
```

```
select E.title, C.name, C.house
from   Characters C, Episodes E, Appearances A
where  C.house = ‘Targaryen’
and    A.name = C.name
and    E.season = A.season
and    E.num = A.num
```

- SPJ queries are the basic, and the most common, kind of a SQL query
- They read from at least one relation
- They return exactly one relation, which may be empty
- Because inputs and outputs are relations, SPJ queries are fully compositional - this is called closure

Game of Thrones

Characters

name	house
Eddard	Stark
Jon Snow	Stark
Tyrion	Lannister
Daenerys	Targaryen

Episodes

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

Appearances

name	season	num
Eddard	1	1
Eddard	1	2
Jon Snow	1	2
Jon Snow	2	1
Jon Snow	2	2
Tyrion	1	1
Tyrion	1	2
Tyrion	2	2
Daenerys	1	2
Daenerys	2	1

```
select E.title, C.name, C.house
from   Characters C, Episodes E, Appearances A
where  C.house = 'Targaryen'
and    A.name = C.name
and    E.season = A.season
and    E.num = A.num
```

Result

title	name	house
The Kingsroad	Daenerys	Targaryen
The North Remembers	Daenerys	Targaryen

Enter relational algebra

- SQL queries are compiled into relational algebra statement

- Formally: the data manipulation aspect of the relational model. Takes relations as input, produces relations as output.
- Practically: a programming language for relational databases
  - simpler and less powerful than a general programming language
  - easier to learn (for us)
  - easier to make efficient (for the database management system)

- Relational algebra is an algebra: relation variables / constants are operands, operators act on such variables and constants

What is an algebra?

- A system consisting of operators and operands
- We are all familiar with the algebra of arithmetic: operators are + – ×, operands are constants, like *42*, or variables, like *x*
- Expressions are made up of operators, operands, optionally grouped by parentheses, e.g., $(x + 3) / (y - 1)$

- In relational algebra:
  - operands are variables that stand for relations
  - constants stand for finite relations (think: a particular set of tuples)
  - let's look at operators

Relational algebra operations

1. The usual set operations: union $\cup$, intersection $\cap$, set difference $\setminus$, but applied to relations (sets of tuples)

2. Operations that remove parts of a relation
   - selection removes rows (tuples)
   - projection removes columns (attributes)

3. Operations that combine tuples of two relations
   - Cartesian product - pairs up tuples in two relations in all possible ways
   - join - selectively pairs up tuples from two relations

4. A renaming operation changes relation schema, re-assigning either relation name or names of attributes

Set operations on relations

Definition: Relations *R* and *S are* *union-compatible* if their schemas define attributes with the same (or compatible) domains.

Set operations can only be applied to union-compatible relations.

R

id	name	age
1	Ann	18
2	Jane	22

S

id	name	age
1	Ann	18
3	Mike	21
4	Dave	27

$R \cup S$

id	name	age
1	Ann	18
2	Jane	22
3	Mike	21
4	Dave	27

$R \cap S$

id	name	age
1	Ann	18

R / S

id	name	age
2	Jane	22

S / R

id	name	age
3	Mike	21
4	Dave	27

Note: (1, Ann, 18) appears only once in the result of $R \cup S$

Selection

The selection operator, applied to relation *R*, produces a new relation with a subsets of *R*'s tuples. Tuples in the new relation are those that satisfy some condition *c*.

$$\boxed{\sigma_C(R)}$$

$Episodes$

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

$\sigma_{viewers>3M} Episodes$

season	num	title	viewers
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

Note: $\sigma_C(R)$ has at most as many rows as *R*

Projection

The projection operator, applied to relation *R*, produces a new relation with a subsets of *R*'s attributes.

$$\boxed{\pi_{A_1, A_2, \ldots, A_n}(R)}$$

$Episodes$

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

$\pi_{season, title} Episodes$

season	title
1	Winter is Coming
1	The Kingsroad
2	The North Remembers
2	The Night Lands

$\pi_{season} Episodes$

season
1
2

Note: $\pi_{A_1, A_2, \ldots, A_n}(R)$ has at most as many rows as *R*

Why not exactly as many?

Cartesian product

The Cartesian product (or cross product) of two relations *R* and *S* is the set of pairs, formed by choosing the first element from *R* and the second element from *S*.

$$\boxed{R \times S}$$

Characters

<u>name</u>	house
Tyrion	Lannister
Daenerys	Targaryen

Episodes

<u>season</u>	<u>num</u>	title
1	1	Winter is Coming
1	2	The Kingsroad

$Characters \times Episodes$

<u>name</u>	house	<u>season</u>	<u>num</u>	title
Tyrion	Lannister	1	1	Winter is Coming
Tyrion	Lannister	1	2	The Kingsroad
Daenery	Targaryen	1	1	Winter is Coming
Daenery	Targaryen	1	2	The Kingsroad

Note: there are exactly $|R| * |S|$ tuples in $R \times S$

Join

The join of two relations R and S is the set of pairs, formed by choosing the first element from R and the second element from S, such that the corresponding tuples in R and S meet some condition c .

$\boxed{R \bowtie_c S}$

Characters

name	house
Tyrion	Lannister
Daenerys	Targaryen

Appearances

name	season	num
Jon Snow	2	1
Tyrion	1	1
Tyrion	2	2
Daenerys	1	2

$Characters \bowtie_{name} Appearances$

name	house	name	season	num
Tyrion	Lannister	Tyrion	1	1
Tyrion	Lannister	Tyrion	2	2
Daenerys	Targaryen	Daenerys	1	2

Note: there are at most $|R| * |S|$ tuples in $R \bowtie_c S$

Join vs. Cartesian product

Conceptually, to compute $R \bowtie_C S$

1. compute a Cartesian product R × S

2. then compute a selection σ_C (R × S) using the join condition

$$\boxed{R \bowtie_C S = \sigma_C (R \times S)}$$

$$R \bowtie_{R.age < S.age} S = \sigma_{R.age < S.age} (R \times S)$$

R.id	R.name	R.age	S.id	S.name	S.age
1	Ann	18	3	Mike	21
1	Ann	18	4	Dave	27
~~2~~	~~Jane~~	~~22~~	~~3~~	~~Mike~~	~~21~~
2	Jane	22	4	Dave	27

Natural join

The natural join of two relations *R* and *S* is a shorthand notation for a join with the condition: the pair up tuples from *R* and *S* join if they agree on the values of the *common attributes*.

$\boxed{R \bowtie S}$

R

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

S

sid	did	cid	term	grade
1111	1	210	Fall 2012	A
2222	1	220	Winter 2013	

$R \bowtie S$

R.sid	R.name	R.gpa	S.sid	S.did	S.cid	S.term	S.grade
1111	Joe	3.2	1111	1	210	Fall 2012	A
2222	Ann	4.0	2222	1	220	Winter 2013	

Note: there are at most $|R| * |S|$ tuples in $R \bowtie_C S$

Renaming

Sometimes it is necessary to rename a relation, or columns in the relation. For this, we use the renaming operator.

Boats (bid, name, color) $\Rightarrow$ *Vessels (vid, handle, shade)*

$$\rho_{Vessels(vid,\ handle,\ shade)}\ (Boats)$$

Boats (bid, name, color) $\Rightarrow$ *Barges (bid, name, color)*

$$\rho_{Barges}\ (Boats)$$

Relational algebra operations (recap)

1. The usual set operations: union $\cup$, intersection $\cap$, set difference $\setminus$, but applied to relations (sets of tuples)

2. Operations that remove parts of a relation

   - selection removes rows (tuples)

   - projection removes columns (attributes)

3. Operations that combine tuples of two relations

   - Cartesian product - pairs tuples in two relations in all possible ways

   - join - selectively pairs tuples from two relations

4. A renaming operation changes relation schema, re-assigning either relation name or names of attributes

Behold the simplicity!

- A limited set of operations
- All operations take relations as input and produce relations as output
- No pointers (i.e., no pointer chasing)

A first-order logic - based formalism, aimed specifically at efficiently processing large datasets in bulk

Combining operations into queries

What are the names of students whose GPA is at least 3.5?

Students

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

1. Select those *Students* tuples that have $gpa \geq 3.5$.
2. Get (project) the value of the *name* column for these tuples.

$$\pi_{name}(\ \sigma_{gpa \geq 3.5}\ (Students))$$

name
Ann
Mike

Combining operations into queries

What are the names of students who got an A in any course?

Students

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

Enrollment

sid	did	cid	term	grade
1111	1	210	Fall 2015	A
2222	1	220	Winter 2016	

1. Join *Students* with *Enrollment* (natural join)
2. Select only those tuples where *grade = 'A'*
3. Project the value of the *name* column for the resulting tuples

$$\pi_{name}(\ \sigma_{grade='A'}(\ Students \bowtie Enrollment))$$

name
Joe

Combining operations into queries

What are the names of students who got an A in any course?

Students

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

Enrollment

sid	did	cid	term	grade
1111	1	210	Fall 2015	A
2222	1	220	Winter 2016	

1. Select tuples in *Enrollment* where *grade = 'A'*
2. Join these tuples with *Students* (natural join)
3. Project the value of the *name* column for the resulting tuples

$$\pi_{name}\,(\,Students \bowtie (\,\sigma_{grade='A'}\;Enrollment))$$

name
Joe

We compute the same result by executing operators in a different order!

Examples

Sailors (sid, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (bid, name, color)

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

Reserves (sid, bid, day)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List names of boats. $\pi_{name}(Boats)$

List ratings and ages sailors. $\pi_{rating,\, age}(Sailors)$

List names of sailors who are over 21 years old.

$$\pi_{name}(\sigma_{age>21}(Sailors))$$

List names of red boats. $\pi_{name}(\sigma_{color=red}(Boats))$

Examples

Sailors (<u>sid</u>, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (<u>bid</u>, name, color)

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

Reserves (<u>sid</u>, <u>bid</u>, <u>day</u>)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List ids of boats named Interlake

List ids of boats reserved on 10/10/12

Examples (solution)

Sailors (<u>sid</u>, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (<u>bid</u>, name, color)

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

Reserves (<u>sid</u>, <u>bid</u>, <u>day</u>)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List ids of boats named Interlake

$$\pi_{bid}(\sigma_{name=Interlake}(Boats))$$

List ids of boats reserved on 10/10/12

$$\pi_{bid}(\sigma_{day=10/10/12}(\mathrm{Reserves}))$$

Examples

Sailors (*sid*, *name*, *rating*, *age*)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (*bid*, *name*, *color*)

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

Reserves (*sid*, *bid*, *day*)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List ids of sailors who reserved boat 102

$$\pi_{sid} (\sigma_{bid=102} \text{Reserves})$$

List names of sailors who reserved boat 102

$$\pi_{name} (\text{Sailors} \bowtie (\sigma_{bid=102} \text{Reserves}))$$

$$\pi_{name} (\sigma_{bid=102} (\text{Sailors} \bowtie \text{Reserves}))$$

both are correct!
which is better?

Examples

Sailors (sid, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (bid, name, color)

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

Reserves (sid, bid, day)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List names of sailors who reserved the red Interlake.

$$\pi_{Sailors.name}\Big(Sailors \bowtie_{S.sid=R.sid} \big((\sigma_{name=Interlake\ and\ color=red}\ Boats) \bowtie Reserves\big)\Big)$$

any other way to do this?

Examples

Sailors (*sid*, *name*, *rating*, *age*)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (*bid*, *name*, *color*)

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

Reserves (*sid*, *bid*, *day*)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List names of boats that were reserved by Horatio.

$$\pi_{Boats.name}\,(\,(\sigma_{Sailors.name=Horatio}\,Sailors) \bowtie_{S.sid=R.sid} (\,Boats \bowtie Reserves))$$

any other way to do this?

Examples

Sailors (<u>sid</u>, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (<u>bid</u>, name, color)

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

Reserves (<u>sid</u>, <u>bid</u>, <u>day</u>)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List days on which some sailor with rating higher than 7 was at sea

Examples (solution)

Sailors (*<u>sid</u>*, *name*, *rating*, *age*)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (*<u>bid</u>*, *name*, *color*)

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

Reserves (*<u>sid</u>*, *<u>bid</u>*, *<u>day</u>*)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List days on which some sailor with rating higher than 7 was at sea

$$\pi_{day}((\sigma_{rating>7}\ Sailors) \bowtie Reserves)$$

Examples

Sailors (sid, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (bid, name, color)

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

Reserves (sid, bid, day)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List names and colors of boats that were reserved by Zorba

Examples (solution)

Sailors (sid, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (bid, name, color)

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

Reserves (sid, bid, day)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

List names and colors of boats that were reserved by Zorba

$$\pi_{Boats.name,\ Boats.color}\Big(((\sigma_{name=Zorba}\ Sailors) \bowtie Reserves) \bowtie_{R.bid=B.bid} Boats) \Big)$$

Renaming

Sometimes it is necessary to rename a relation, or columns in the relation. For this, we use the renaming operator.

Boats (bid, name, color) $\Rightarrow$ *Vessels (vid, handle, shade)*

$$\rho_{Vessels(vid,\ handle,\ shade)}\ (Boats)$$

Boats (bid, name, color) $\Rightarrow$ *Barges (bid, name, color)*

$$\rho_{Barges}\ (Boats)$$

A self-join

A self-join is a join that joins together tuples from two copies of the same table

List all possible heterosexual couples (girl name, boy name), where the boy is older than the girl.

People

id	name	age	gender
1	Ann	18	F
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

$$\pi_{Girls.name,\, Boys.name} \Big(\rho_{Girls} \big(\sigma_{gender=F}\, People \big) \bowtie_{Girls.age<Boys.age} \rho_{Boys} \big(\sigma_{gender=M}\, People \big) \Big)$$

Relational algebra - recap

- Is the data manipulation aspect of the relational model
- Expressions are procedural, in that the order of operators is specified explicitly
- Often several different relational algebra expressions will give the same result (for all valid instances!), but will have different performance characteristics
- SQL can be seen as a declarative implementation of relational algebra

Outline

- Part 1: Relational algebra
- Part 2: SQL
  - Operations on a single relation
  - Combining relations
  - Aggregation
  - Additional SQL features

SQL

- SQL (“seekwel”) stands for Structured Query Language
- Made up of 2 parts:
  - Data Definition Language (DDL) - used to create or modify a relational schema
  - Data Manipulation Language (DML) - used to retrieve or modify data in a schema
  - We call SQL statements queries

SQL, relational algebra, do we need both?

Characters

<u>name</u>	house
Eddard	Stark
Jon Snow	Stark
Tyrion	Lannister
Daenerys	Targaryen

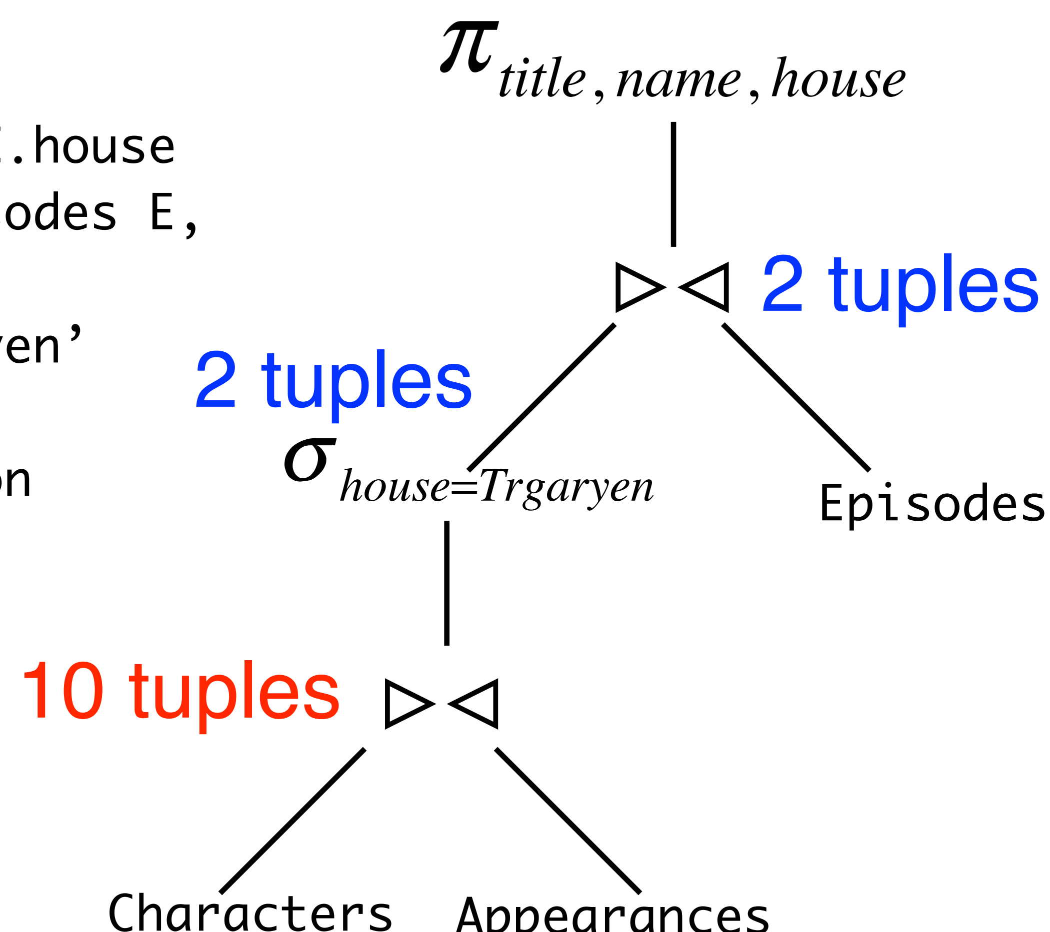
Episodes

<u>season</u>	<u>num</u>	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

Appearances

<u>name</u>	<u>season</u>	<u>num</u>
Eddard	1	1
Eddard	1	2
Jon Snow	1	2
Jon Snow	2	1
Jon Snow	2	2
Tyrion	1	1
Tyrion	1	2
Tyrion	2	2
Daenerys	1	2
Daenerys	2	1

```
select E.title, C.name, C.house
from   Characters C, Episodes E,
       Appearances A
where  C.house = 'Targaryen'
and    A.name = C.name
and    E.season = A.season
and    E.num = A.num
```



SQL, relational algebra, do we need both?

Characters

name	house
Eddard	Stark
Jon Snow	Stark
Tyrion	Lannister
Daenerys	Targaryen

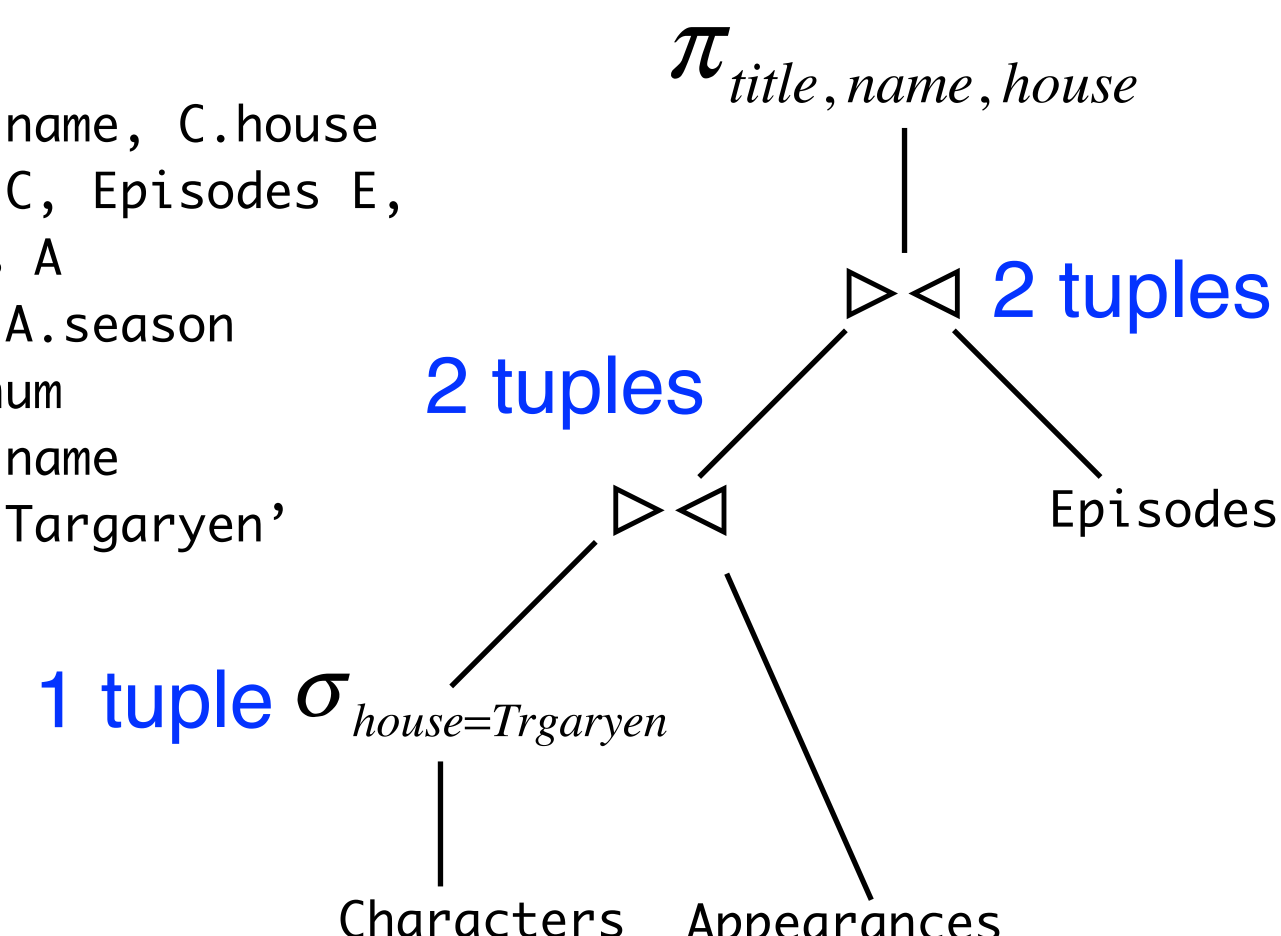
Episodes

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

Appearances

name	season	num
Eddard	1	1
Eddard	1	2
Jon Snow	1	2
Jon Snow	2	1
Jon Snow	2	2
Tyrion	1	1
Tyrion	1	2
Tyrion	2	2
Daenerys	1	2
Daenerys	2	1

```
select E.title, C.name, C.house
from   Characters C, Episodes E,
       Appearances A
where  E.season = A.season
and    E.num = A.num
and    A.name = C.name
and    C.house = 'Targaryen'
```



SQL, relational algebra, do we need both?

Characters

name	house
Eddard	Stark
Jon Snow	Stark
Tyrion	Lannister
Daenerys	Targaryen

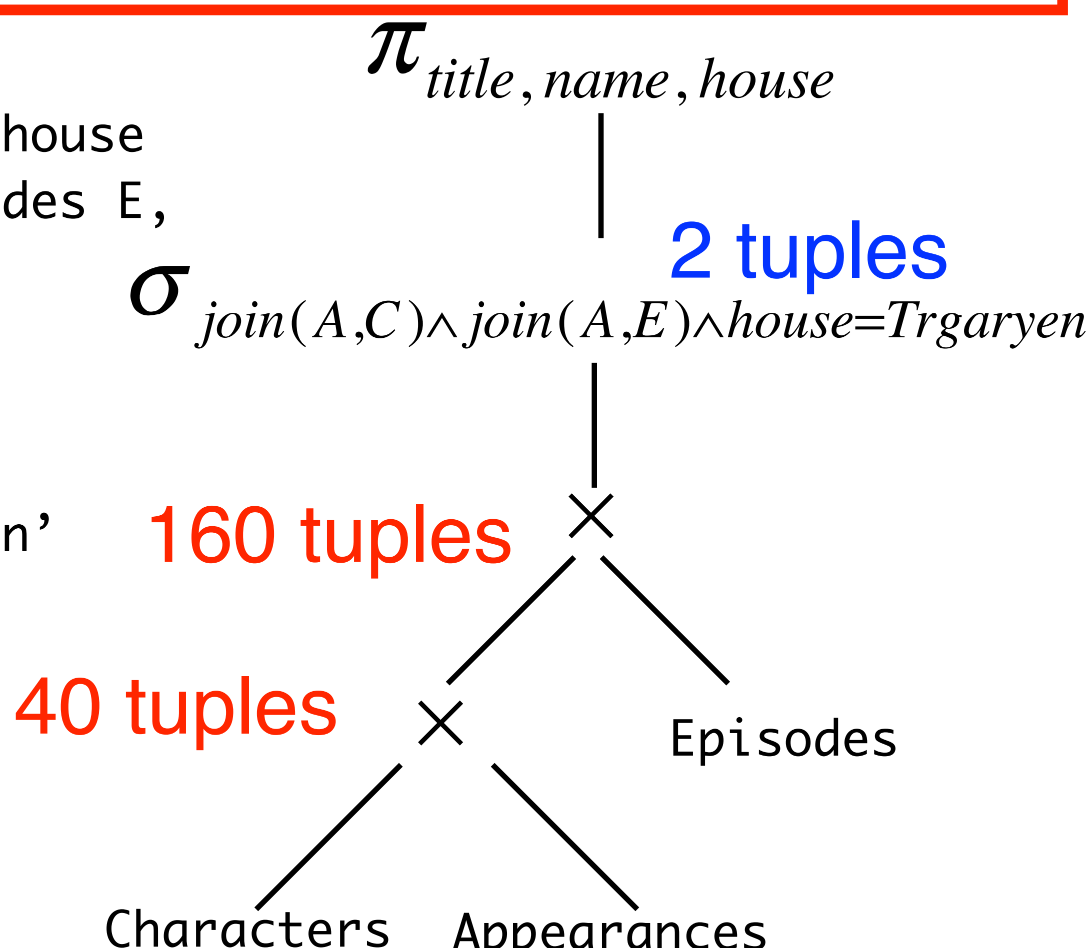
Episodes

season	num	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

Appearances

name	season	num
Eddard	1	1
Eddard	1	2
Jon Snow	1	2
Jon Snow	2	1
Jon Snow	2	2
Tyrion	1	1
Tyrion	1	2
Tyrion	2	2
Daenerys	1	2
Daenerys	2	1

```
select E.title, C.name, C.house
from   Characters C, Episodes E,
       Appearances A
where  E.season = A.season
and    E.num = A.num
and    A.name = C.name
and    C.house = 'Targaryen'
```

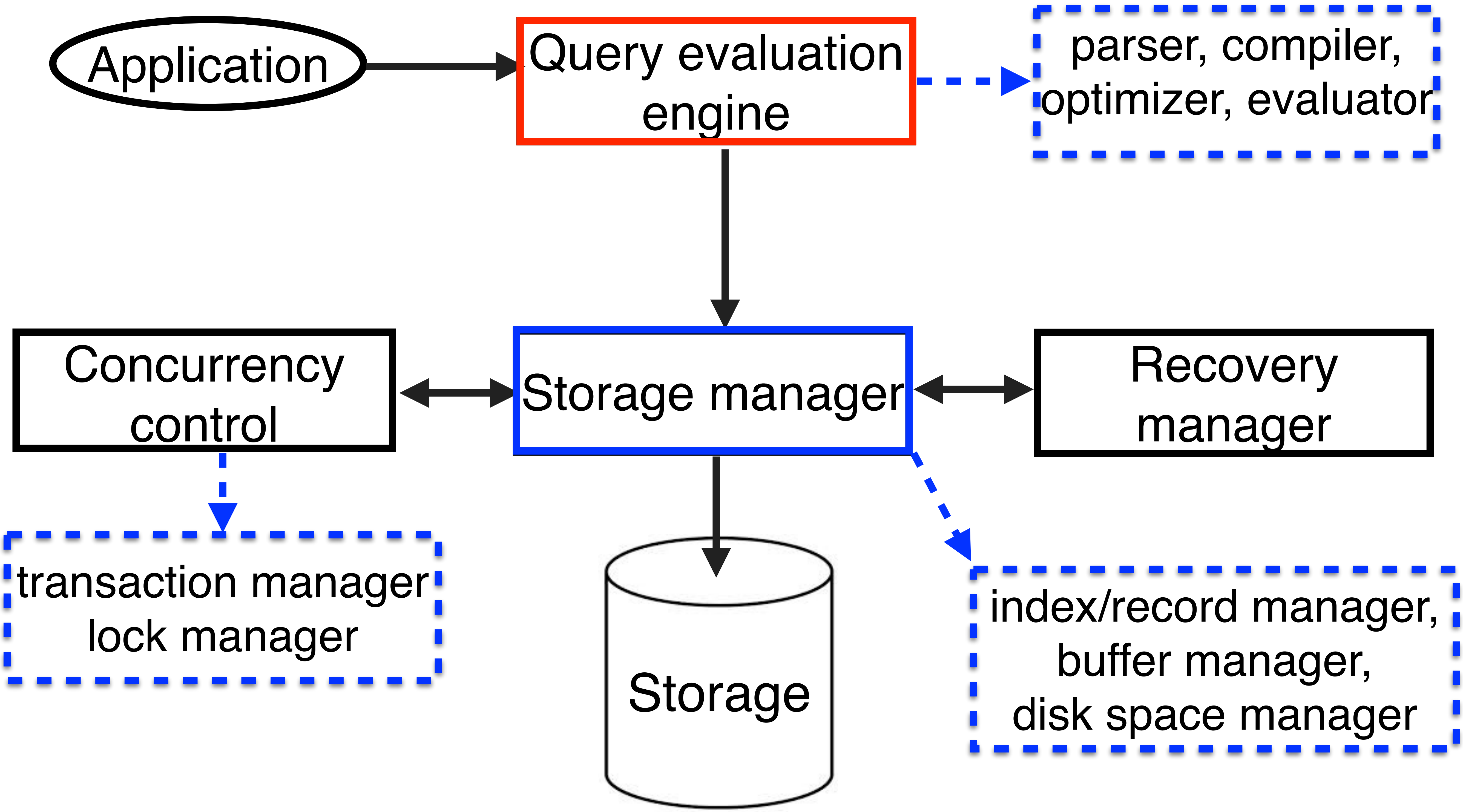


How do we pick a good rewriting?

- Good question! The answer: we don't!

- Rewriting SQL queries into relational algebra statements (or some similar representation) is the job of the query optimizer
- We, people, do what we do well: specify **what** we want to compute using SQL
- The system does what it does well: decides on an efficient query evaluation procedure - **how** to compute the result
- I got one word for you -   declarativeness!

Architecture of a typical DBMS



Overview of query optimization

Given a SQL query, there may be different execution plans that produce the same result but that have different performance characteristics

- Goal of query optimization: find an efficient query execution plan for a given query
- Query execution plan is represented by a tree of relational algebra operators, annotated with a choice of an algorithm for each operator
- The optimizer also decides how to access data in a particular relation, i.e., which access path to use

Operations on a single relation

R

id	name	age	gender
1	Ann	18	F
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

Given a relation *R*, return a new relation *R'* that contains a sub-set of the rows / columns from *R*

Selection in SQL

$$\boxed{\sigma_C(R)}$$

```
select *
from   R
where  C
```

R

id	name	age	gender
1	Ann	18	F
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

$$\sigma_{age \geq 21}(R)$$

```
select *
from   R
where  age >= 21
```

id	name	age	gender
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

$$\sigma_{age \geq 21\ AND\ gender='M'}(R)$$

```
select *
from   R
where  age >= 21
and    gender ='M'
```

id	name	age	gender
3	Mike	21	M
4	Dave	27	M

Projection in SQL

$$\pi_{A_1, A_2, \ldots, A_n}(R)$$

```
select A1, A2, ..., An
from   R
```

R

id	name	age	gender
1	Ann	18	F
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

$$\pi_{id,\, name}(R)$$

```
select id, name
from   R
```

id	name
1	Ann
2	Jane
3	Mike
4	Dave

$$\pi_{gender}(R)$$

gender
F
M

```
select gender
from   R
```

gender
F
F
M
M

SQL queries compute bags, not sets!

- A bag (aka a multi-set) is an unordered collection with duplicates
- This is unlike a set, which is also unordered, but there are no duplicates
- Relational model, relational algebra work with sets
- SQL works with bags
- Why bags: efficiency of implementation is the main reason

R

id	name	age	gender
1	Ann	18	F
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

```
select gender
from   R
```

gender
F
F
M
M

```
select distinct gender
from   R
```

gender
F
M

We use distinct to covert from a bag to a set

Combining selection and projection

$$\pi_{A_1, A_2, \ldots, A_n}(\sigma_C(R))$$

```
select A1, A2, ..., An
from   R
where  C
```

What are the names of students whose GPA is at least 3.5?

Students

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

$$\pi_{name}(\sigma_{gpa \geq 3.5}(Students))$$

```
select name
from   Students
where  gpa >= 3.5
```

name
Ann
Mike

Examples

Sailors (*sid*, *name, rating, age*)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (*bid*, *name, color*)

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

Reserves (*sid*, *bid*, *day*)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

(a) List names of boats.

(b) List ratings and ages sailors.

(c) List names of sailors who are over 21 years old.

(d) List names of red boats.

(e) List ids of boats that have ever been reserved.

String expressions

Sailors (sid, name, rating, age) *Boats (bid, name, color)* *Reserves (sid, bid, day)*

Find name, age, 2*rating of all sailors whose name starts with A or contains 'ust' as a substring starting from position 2

Q1 in Query.sql

```
select name, age, rating * 2 as twice_the_rating
from   Sailors
where  name like 'A%' or name like '_ust%';
```

`as` is a way to name a field in the result

`%` denotes 0 or more arbitrary characters

`_` denotes exactly 1 arbitrary character

`like` is used for string matching

Expressions and strings: examples

Sailors (<u>sid</u>, name, rating, age) *Boats (<u>bid</u>, name, color)* *Reserves (<u>sid</u>, <u>bid</u>, <u>day</u>)*

Find names, sids of sailors whose rating * 5 is less than their age

Q2

```
select name, sid
from   Sailors
where  rating * 5 < age
```

Find names, sids of sailors whose names contain the letter *u*

Q3

```
select name, sid
from   Sailors
where  name like '%u%'
```

Null values

- null stands for a missing / unknown / inapplicable value
- SQL provides special comparison operators for null (cannot use = < > !=)

Sailors (sid, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius	*null*	25

Q4

```
select *
from Sailors
where rating is not null;
```

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18

Q5

```
select *
from Sailors
where rating is null;
```

sid	name	rating	age
5	Julius	*null*	25

Null values (II)

- null stands for a missing / unknown / inapplicable value
- SQL provides special comparison operators for null (cannot use = < > !=)

Sailors (<u>sid</u>, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius	*null*	25

Q6

```
select sid, rating * 0
from   Sailors;
```

sid	rating
1	0
2	0
3	0
4	0
5	*null*

The value of rating * 0 for sid = 5 is null, not 0!

Renaming attributes

Q7

```
select sid as sailor_id, rating / 10 as normalized_rating
from   Sailors;
```

the *as* keyword is optional

```
select sid sailor_id, rating / 10 normalized_rating
from   Sailors;
```

sailor_id	normalized_rating
1	0.7
2	1
3	0.5
4	0.8
5	*null*

Renaming relations

Q8

```
select S.sid, S.rating / 10 as normalized_rating
from   Sailors S;
```

sid	normalized_rating
1	0.7
2	1
3	0.5
4	0.8
5	*null*

Sorting results

- Sets and bags are unordered collections
- Sometimes we want to see results in sorted order, an *order by* clause allows us to do this

```
select   Ai, Aj, ..., An
from     R
where    C
order by Ak [desc], Al [desc], ..., Am [desc]
```

1. compute the result of $\pi_{A_i, A_j, \ldots, A_n}(\sigma_C(R))$
2. sort the result by A_k, break ties by A_l, ..., break ties by A_m
3. the optional `desc` keyword after each column specifies descending (higher to lower) sort order; the default order is ascending (lower to higher)

Sorting results: example

Retrieve all reservations sorted by sid, with ties broken by bid

Q9

```
select *
from Reserves
order by sid, bid
```

sid	bid	day
1	101	10/10/12
1	101	10/7/12
1	102	10/10/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

Retrieve all reservations sorted by sid (descending), with ties broken by bid (ascending)

Q10

```
select *
from Reserves
order by sid desc, bid
```

sid	bid	day
4	103	19/9/12
3	101	7/11/12
3	102	7/8/12
2	102	11/9/12
2	102	7/11/12
1	101	10/10/12
1	101	10/7/12
1	102	10/10/12

Examples

Sailors (sid, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius		25

Boats (bid, name, color)

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

Reserves (sid, bid, day)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

(a) Find the color of a boat named clipper.

(b) Find all sailors who are 35 or older, sort them by name.

(c) List the names of red boats, sorted by boat id.

(d) Find all boats that are either red or called Interlake.

Combining relations

R

id	name	age	gender
1	Ann	18	F
2	Jane	22	F
3	Mike	21	M
4	Dave	27	M

S

id	major	gpa
1	Math	3.7
2	CS	3.8
3	English	3.8
4	Philosophy	3.2

Informally: given relations *R, S,* return a new relation *T* that contains some combination of tuples from *R, S*

Cartesian product

List all pairs of Girls (G) and Boys (B)

G

id	name	age
1	Ann	18
2	Jane	22

B

id	name	age
3	Mike	21
4	Dave	27

```
select *
from G, B
```

$$G \times B$$

G.id	G.name	G.age	B.id	B.name	B.age
1	Ann	18	3	Mike	21
1	Ann	18	4	Dave	27
2	Jane	22	3	Mike	21
2	Jane	22	4	Dave	27

List all pairs of Girls (G) and Boys (B) that involve Ann

```
select *
from   G, B
where  G.name = 'Ann'
```

$$\sigma_{G.name='Ann'}(G \times B)$$

G.id	G.name	G.age	B.id	B.name	B.age
1	Ann	18	3	Mike	21
1	Ann	18	4	Dave	27

Cartesian product

List all pairs of Girls (G) and Boys (B) where the girl is younger than the boy

G

id	name	age
1	Ann	18
2	Jane	22

B

id	name	age
3	Mike	21
4	Dave	27

$$\sigma_{G.age<B.age}\,(G \times B)$$

```
select *
from  G, B
where G.age < B.age
```

G.id	G.name	G.age	B.id	B.name	B.age
1	Ann	18	3	Mike	21
1	Ann	18	4	Dave	27
2	Jane	22	4	Dave	27

or is this a join?

$$G \bowtie_{G.age<B.age} B = \sigma_{G.age<B.age}\,(G \times B)$$

yes, both are right, both are expressed by the same SQL query!

Join

Students

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

Enrollment

sid	did	cid	term	grade
1111	1	210	Fall 2012	A
2222	1	220	Winter 2013	*null*

List the name, GPA and enrollment information for all students who have been enrolled in a course.

Students ⋈ *Enrollment*

S.name	S.gpa	E.did	E.cid	E.term
Joe	3.2	1	210	Fall 2012
Ann	4	1	220	Winter 2013

```
select S.name, S.gpa, E.did, E.cid, E.term
from   Students S, Enrollment E
where  S.sid = E.sid
```

Note that renaming relations is handy here

Join

Students

sid	name	gpa
1111	Joe	3.2
2222	Ann	4.0
3333	Mike	3.5

Enrollment

sid	did	cid	term	grade
1111	1	210	Fall 2012	A
2222	1	220	Winter 2013	*null*

List the name, GPA and enrollment information for all students who have been enrolled in a course and have a grade for that course.

$$Students \bowtie \sigma_{E.grade\ is\ not\ null}(Enrollment)$$

$$\sigma_{E.grade\ is\ not\ null}(Students \bowtie Enrollment)$$

S.sid	E.did	E.cid	E.term
1111	1	210	Fall 2012
2222	1	220	Winter 2013

```
select S.name, S.gpa, E.did, E.cid, E.term
from   Students S, Enrollment E
where  S.sid = E.sid
and    E.grade is not null
```

Note that the same SQL query corresponds to both relational algebra statements.

SQL is declarative

$Students \bowtie \sigma_{E.grade\ is\ not\ null}\ (Enrollment)$

$\sigma_{\ E.grade\ is\ not\ null}\ (Students \bowtie Enrollment)$

```
select S.name, S.gpa, E.did, E.cid, E.term
from   Students S, Enrollment E
where  S.sid = E.sid
```

- A relational algebra statement specifies the order of operations
  - the first statement first selects and then joins, while the second first joins and then selects
  - in a sense, we have to specify both what to compute and how to compute it
- Both relational algebra statements return the same result, but the processing costs differ
- SQL queries are declarative: we only say what to compute not how
- This makes it easier for us, and leads to a more efficient query execution plan (because the database engine knows best)

More examples

Sailors (<u>sid</u>, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius	*null*	25

Boats (<u>bid</u>, name, color)

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

Reserves (<u>sid</u>, <u>bid</u>, <u>day</u>)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

Find names of sailors who have reserved boat 101

Q11

```
select S.name
from   Sailors S, Reserves R
where  S.sid = R.sid
and    R.bid = 101
```

name
Dustin
Dustin
Horatio

More examples

Sailors (sid, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius	*null*	25

Boats (bid, name, color)

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

Reserves (sid, bid, day)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

Find ages of sailors who have reserved a red boat

Q12

```
select S.age
from   Sailors S, Reserves R, Boats B
where  S.sid = R.sid
and    R.bid = B.bid
and    B.color = 'red'
```

age
45
35
35
35

More examples

Sailors (<u>sid</u>, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius	*null*	25

Boats (<u>bid</u>, name, color)

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

Reserves (<u>sid</u>, <u>bid</u>, <u>day</u>)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

Find unique ages of sailors who have reserved a red boat

Q13

```
select distinct S.age
from   Sailors S, Reserves R, Boats B
where  S.sid = R.sid
and    R.bid = B.bid
and    B.color = 'red'
```

age
45
35

How do we read a SQL query?

```
select   S.name, B.name, R.day
from     Sailors S, Reserves R, Boats B
where    S.sid = R.sid
and      R.bid = B.bid
and      B.color = ‘green’
order by S.name, R.day
```

- First, look at the from clause - which tables are being used
- Next, look at the where where clause - what are the selection and join conditions
- Then, look at the select clause - which columns are used in the result
- Finally, if required, think about the order in which results are returned

To write queries, think in the same order: from, where, select, order by.

More examples

Sailors (sid, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius	*null*	25

Boats (bid, name, color)

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

Reserves (sid, bid, day)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

Find unique names and ratings of sailors who have reserved a boat named Interlake and whose name contains the letter ‘u’.

Q14*

Find days of reservations made for a red boat by a sailor who is under 40 years old.

Q15*

More examples

Sailors (sid, name, rating, age)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius	*null*	25

Boats (bid, name, color)

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

Reserves (sid, bid, day)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

Find unique names and ratings of sailors who have reserved a boat named Interlake and whose name contains the letter 'u'.

Q14

name	rating
Dustin	7
Rusty	10

More examples

Sailors (*sid*, *name*, *rating*, *age*)

sid	name	rating	age
1	Dustin	7	45
2	Rusty	10	35
3	Horatio	5	35
4	Zorba	8	18
5	Julius	*null*	25

Boats (*bid*, *name*, *color*)

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

Reserves (*sid*, *bid*, *day*)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

Find days of reservations made for a red boat by a sailor who is under 40 years old.

Q15

day
11/9/12
7/11/12
7/8/12

Union, Intersect, Except

Find unique sids and names of sailors who have reserved a red or a blue boat

Q16

```
select distinct S.sid, S.name
from   Sailors S, Boats B, Reserves R
where  S.sid = R.sid  and B.bid = R.bid
and    (B.color = 'red' or B.color = 'blue')
```

Find unique sids and names of sailors who have reserved a red and a blue boat

Q17

```
select distinct S.sid, S.name
from   Sailors S, Boats B1, Reserves R1,
                  Boats B2, Reserves R2
where  S.sid = R1.sid and B1.bid = R1.bid
and    S.sid = R2.sid and B2.bid = R2.bid
and    B1.color = 'red'
and    B2.color = 'blue'
```

Difficult to understand, is there an easier way?

Union, Intersect, Except

Find unique sids, names of sailors who have reserved a red or a blue boat

Option 1

Q16

```
select distinct S.sid, S.name
from    Sailors S, Boats B, Reserves R
where   S.sid = R.sid  and B.bid = R.bid
and     (B.color = 'red' or B.color = 'blue')
```

Option 2

Q18

or

```
select S.sid, S.name
from   Sailors S, Boats B, Reserves R
where  S.sid = R.sid and    B.bid = R.bid
and    B.color = 'red'
UNION
select S.sid, S.name
from   Sailors S, Boats B, Reserves R
where  S.sid = R.sid and    B.bid = R.bid
and    B.color = 'blue';
```

note no "distinct" in Q18:
UNION eliminates duplicates

Union, Intersect, Except

Find unique sids, names of sailors who have reserved a red and a blue boat

Option 1

Q17

```
select distinct S.sid, S.name
from   Sailors S, Boats B1, Reserves R1,
                  Boats B2, Reserves R2
where  S.sid = R1.sid and B1.bid = R1.bid
and    S.sid = R2.sid and B2.bid = R2.bid
and    B1.color = 'red'
and    B2.color = 'blue';
```

Option 2

Q19

and

```
select S.sid, S.name
from   Sailors S, Boats B, Reserves R
where  S.sid = R.sid and    B.bid = R.bid
and    B.color = 'red'
INTERSECT
select S.sid, S.name
from   Sailors S, Boats B, Reserves R
where  S.sid = R.sid and    B.bid = R.bid
and    B.color = 'blue';
```

note no "distinct" in Q19:
UNION eliminates duplicates

Union, Intersect, Except

Find unique sids, names of sailors who have reserved a red but not a blue boat

Option 1: involves correlated subqueries, will see later

Option 2

Q20

but not

```
select S.sid, S.name
from   Sailors S, Boats B, Reserves R
where  S.sid = R.sid and    B.bid = R.bid
and    B.color = 'red'
EXCEPT
select S.sid, S.name
from   Sailors S, Boats B, Reserves R
where  S.sid = R.sid and    B.bid = R.bid
and    B.color = 'blue';
```

note no “distinct" in Q20:
EXCEPT eliminates duplicates

To use set operators...

... we must make sure that the sets involved in the UNION / INTERSECT / EXCEPT are union-compatible, i.e., tuples have the same number of fields, and fields are of compatible types.

```
select S.sid, S.name from Sailors S
UNION
select R.sid, R.day from Reserves R;
```

```
ERROR:  UNION types character varying and date cannot be matched
LINE 3: select R.sid, R.day from Reserves R;
                      ^
```

More examples

Find unique names, bids of boats that have been reserved by sailors with rating > 7

Q21

```
select distinct B.name, B.bid
from    Boats B, Reserves R, Sailors S
where   B.bid = R.bid
and     R.sid = S.sid
and     S.rating > 7
```

Find unique names, bids of boats that have been reserved by sailors with rating > 7 or rating < 5

```
select distinct B.name, B.bid
from    Boats B, Reserves R, Sailors S
where   B.bid = R.bid
and     R.sid = S.sid
and     (S.rating > 7 or S.rating < 5)
```

Q22

```
select B.name, B.bid
from   Boats B, Reserves R, Sailors S
where  B.bid = R.bid
and    R.sid = S.sid
and    S.rating > 7
UNION
select B.name, B.bid
from   Boats B, Reserves R, Sailors S
where  B.bid = R.bid
and    R.sid = S.sid
and    S.rating < 5
```

Q23

note no “distinct" in Q23:
UNION eliminates duplicates

More examples

Find names and bids of boats that have been reserved by sailors with rating > 7 and rating < 5

Q24*

Find names, bids of boats that have been reserved by sailors with rating > 7 but not rating < 5

Q25*

More examples

Find bids of boats that have been reserved by sailors with rating > 7 and rating < 5

Q24

Find bids of boats that have been reserved by sailors with rating > 7 but not rating < 5

Q25

Nested queries

- Sometimes, it is necessary to embed (i.e., nest) a query within a query (usually in the *from* or in the *where* clause)
- This is needed when we have to refer to a value that must itself be computed
- The embedded query is usually called a subquery
- You can nest queries within nested queries within nested queries....

Nested queries

Find names of sailors who have reserved boat 103

Option 1

Q26

```
select S.name
from   Sailors S, Reserves R
where  S.sid = R.sid
and    R.bid = 103
```

Option 2

Q27

```
select S.name
from   Sailors S
where S.sid in (select  R.sid
                from    Reserves R
                where   R.bid = 103)
```

- subquery computes sids of sailors who reserved boat 103
- top-level query gets names of sailors whose sids are in this (multi)set

Nested queries: examples

Find names of sailors who have reserved a red boat

Option 1

Q28

```
select S.name
from   Boats B, Reserves R, Sailors S
where  B.bid = R.bid
and    R.sid = S.sid
and    B.color = 'red'
```

Option 2

Q29

```
select S.name
from   Sailors S
where  S.sid in (select R.sid
                 from   Reserves R
                 where  R.bid in (select B.bid
                                  from   Boats B
                                  where  B.color = 'red'))
```

Nested queries: more examples

Write two queries that compute this: one nested query and one using join.

Find unique colors of boats that were reserved by sailors with rating >7.

Q30*

Find unique names of boats that were reserved during the month of October.

(to get reservations made during the month of October: select * from Reserves where to_char(day, 'Mon') = 'Oct')

Q31*

Nested queries: more examples

Write two queries that compute this: one nested query and one using join.

Find unique colors of boats that were reserved by sailors with rating >7.

Q30.a

Q30.b

Nested queries: more examples

Write two queries that compute this: one nested query and one using join.

Find unique names of boats that were reserved during the month of October.

(to get reservations made during the month of October: select * from Reserves where to_char(day, 'Mon') = 'Oct')

Q31.a

Q31.b

Nested queries: examples

Find names of sailors who have never reserved a red boat

Q32

```
select S.name
from   Sailors S
where  S.sid not in (select R.sid
                     from   Reserves R
                     where  R.bid in (select B.bid
                                      from   Boats B
                                      where  B.color = 'red'))
```

What does this query compute?

Q33

```
select S.name
from   Sailors S
where  S.sid in (select R.sid
                 from   Reserves R
                 where  R.bid not in (select B.bid
                                      from   Boats B
                                      where  B.color = 'red'))
```

Aggregation

- In addition to retrieving data, we can use SQL to summarize / help analyze it.
- SQL supports operations for computing *aggregate values* of any column A
  - COUNT ([DISTINCT] A) - the number of (unique) values in the A column
  - SUM ([DISTINCT] A) - the sum of all (unique) values in the A column
  - AVG ([DISTINCT] A) - the average of all (unique) values in the A column
  - MIN (A) - the minimum value in the A column
  - MAX (A) - the maximum value in the A column

Aggregation: examples

Find the average age of sailors with a rating of 10

Q37
```
select AVG(age)
from   Sailors
where  rating = 10
```

Find the number of sailors

Q38
```
select COUNT(*)
from Sailors
```
think of * as shorthand for all columns

Find the number of sailors, their average age, and minimum rating

Q39
```
select COUNT(*), AVG(age), MIN(rating)
from Sailors
```

Aggregation: more examples

Find the name and age of the oldest sailor

Take 1 (doesn't work!)

```
select name, MAX(age) as age
from   Sailors
```

Rule 1: a query that uses aggregate operators must use only aggregate operators in the select clause, unless the query contains a *group by* clause (we'll see this next).

Take 2 (works)

Q40
```
select S1.name, S1.age
from   Sailors S1
where  S1.age = (select max(S2.age) from Sailors S2)
```

this is also an example of a nested query in the *where* clause

Aggregation: more examples

Count the number of sailors

```
select COUNT(*)
from   Sailors
```

Q38

Count the number of sailor ages

```
select COUNT(age)
from   Sailors
```

Count the number of unique sailor ages

```
select COUNT(distinct age)
from   Sailors
```

Count the number of sailor ages, and of unique sailor ages

```
select COUNT(distinct age) num_unique_ages, COUNT(age) num_ages
from   Sailors
```

Q41

What about rating, rather than age?

Aggregation: more examples

Find the number of sailors who are older than the oldest sailor with the rating of 10

Q42*

List sids, names of sailors with an above average rating

Q43*

Aggregation: more examples

Find the number of sailors who are older than the oldest sailor with the rating of 10

Q42

List sids, names of sailors with an above average rating

Q43

Group-by and having clauses

Find the age of the youngest sailor for each rating level

```
select MIN(age)
from   Sailors
where  rating = 1
```

```
select MIN(age)
from   Sailors
where  rating = 2
```

```
select MIN(age)
from   Sailors
where  rating = 3
```

.....

```
select MIN(age)
from   Sailors
where  rating = 8
```

```
select MIN(age)
from   Sailors
where  rating = 9
```

```
select MIN(age)
from   Sailors
where  rating = 10
```

Group-by and having clauses

Find the age of the youngest sailor for each rating level

Q44

```
select    rating, MIN(age)
from      Sailors
group by  rating
```

Rule 1: a query that uses aggregate operators must use only aggregate operators in the select clause, unless the query contains a *group by* clause.

Rule 2: all non-aggregate columns in the target-list must appear in the *group by* clause

Group-by and having clauses

Find the age of the youngest sailor for each rating level

Q44

```
select    rating, MIN(age)
from      Sailors
group by  rating
```

Find the age of the youngest sailor for each rating level, such that there are at least two sailors with that rating

Q45

```
select    rating, MIN(age)
from      Sailors
group by  rating
having    count(*) > 1
```

Group-by and having clauses

Rule 1: a query that uses aggregate operators must use only aggregate operators in the select clause, unless the query contains a *group by* clause.

Rule 2: all non-aggregate columns in the *select* clause must appear in the *group by* clause

Rule 3: only non-aggregate columns used in the *group by* may appear as non-aggregate columns in the *having* clause.

But we usually only use aggregate columns in the *having* clause anyway.

Aggregation: more examples

Find the number of reservations for each boat that was reserved at least 3 times.

Q46*

Find the bid, name of each red boat that has been reserved, along with an average rating and minimum age of sailors who have reserved that boat.

Q47*

Aggregation: more examples

Find the number of reservations for each boat that was reserved at least 3 times.

Q46

Find the bid, name of each red boat that has been reserved, along with an average rating and minimum age of sailors who have reserved that boat.

Q47

Aggregation: more examples

Find the number of boat reservations by each sailor who is at least 20 years old.

Q48*

Find the number of boat reservations by each sailor who has made at least 2 reservations.

Q49*

Aggregation: more examples

Find the number of boat reservations by each sailor who is at least 20 years old.

Q48

Find the number of boat reservations by each sailor who has made at least 2 reservations.

Q49

Sorting results: more example

Retrieve all information about reservations, sorted by bid

Q50

```
select *
from Reserves
order by bid
```

Retrieve all information about reservations, sorted by bid (descending), with ties broken by sid (ascending)

Q51

```
select *
from Reserves
order by bid desc, sid
```

Retrieve bids of all reserved boats, along with the number of times they were reserved, and the average sailor rating, sorted by average sailor rating

Q52

```
select   R.bid, count(*) num_reservations, avg(S.rating) avg_rating
from     Reserves R, Sailors S
where    R.sid = S.sid
group by R.bid
order by avg_rating
```

Additional SQL features

- Outer joins
- Database modifications
- Views

Running example

Animals (<u>aid</u>, name, hid)

aid	name	hid
1	elephant	101
2	giraffe	101
3	whale	102
4	dolphin	102
5	okapi	null

Habitats (<u>hid</u>, name)

hid	name
101	African savanna
102	North Pacific ocean
103	Eurasian tundra

List names of animals and names of their habitats

Q60

```
select A.name as animal_name,
       H.name as habitat_name
from   Animals A, Habitats H
where  A.hid = H.hid
```

animal_name	habitat_name
elephant	African savanna
giraffe	African savanna
whale	North Pacific ocean
dolphin	North Pacific ocean

Left outer join

Animals (aid, name, hid)

aid	name	hid
1	elephant	101
2	giraffe	101
3	whale	102
4	dolphin	102
5	okapi	null

Habitats (hid, name)

hid	name
101	African savanna
102	North Pacific ocean
103	Eurasian tundra

List names of animals and, if their habitat is known, also list their habitat.

Q61

```
select A.name as animal_name,
       H.name as habitat_name
from   Animals A left outer join Habitats H
on     (A.hid = H.hid)
```

animal_name	habitat_name
elephant	African savanna
giraffe	African savanna
whale	North Pacific ocean
dolphin	North Pacific ocean
okapi	null

Right outer join

Animals (aid, name, hid)

aid	name	hid
1	elephant	101
2	giraffe	101
3	whale	102
4	dolphin	102
5	okapi	null

Habitats (hid, name)

hid	name
101	African savanna
102	North Pacific ocean
103	Eurasian tundra

List names of habitats and, if there are animals inhabiting them, also list names of these animals.

Q62

```
select A.name as animal_name,
       H.name as habitat_name
from   Animals A right outer join Habitats H
on     (A.hid = H.hid)
```

animal_name	habitat_name
elephant	African savanna
giraffe	African savanna
whale	North Pacific ocean
dolphin	North Pacific ocean
null	Eurasian tundra

Full outer join

Animals (aid, name, hid)

aid	name	hid
1	elephant	101
2	giraffe	101
3	whale	102
4	dolphin	102
5	okapi	null

Habitats (hid, name)

hid	name
101	African savanna
102	North Pacific ocean
103	Eurasian tundra

List names of animals and names of habitats where they live. If no habitat is known for an animal, or no animal is known to live in a habitat - list these anyway.

Q63

```
select A.name as animal_name,
       H.name as habitat_name
from   Animals A full outer join Habitats H
on     (A.hid = H.hid)
```

animal_name	habitat_name
elephant	African savanna
giraffe	African savanna
whale	North Pacific ocean
dolphin	North Pacific ocean
null	Eurasian tundra
okapi	null

Join vs. outer join

Animals (aid, name, hid)

A.aid	A.name	A.hid
1	elephant	101
3	whale	102
5	okapi	null

Habitats (hid, name)

H.hid	H.name
101	African savanna
102	North Pacific ocean
103	Eurasian tundra

```
select *
from   Animals A, Habitats H
where  A.hid = H.hid
```

$$\sigma_{A.hid=H.hid}(Animals \times Habitats)$$

A.aid	A.name	A.hid	H.hid	H.name
1	elephant	101	101	African savanna
~~1~~	~~elephant~~	~~101~~	~~102~~	~~North Pacific ocean~~
~~1~~	~~elephant~~	~~101~~	~~103~~	~~Eurasian tundra~~
~~3~~	~~whale~~	~~102~~	~~101~~	~~African savanna~~
3	whale	102	102	North Pacific ocean
~~3~~	~~whale~~	~~102~~	~~103~~	~~Eurasian tundra~~

Join vs. outer join

Animals (<u>aid</u>, name, hid)

A.aid	A.name	A.hid
1	elephant	101
3	whale	102
5	okapi	null

Habitats (<u>hid</u>, name)

H.hid	H.name
101	African savanna
102	North Pacific ocean
103	Eurasian tundra

```
select *
from   Animals A full outer join Habitats H
on     A.hid = H.hid
```

A.aid	A.name	A.hid	H.hid	H.name
1	elephant	101	101	African savanna
3	whale	102	102	North Pacific ocean
5	okapi	null	null	null
null	null	null	103	Eurasian tundra

join

left outer

right outer

Examples

Q64

```
select A.name as animal_name, H.name as habitat_name
from   Animals A left outer join Habitats H
on     (A.hid = H.hid)
where  A.name like '%o%'
```

Q65

```
select H.name as habitat_name, count(*)
from   Animals A full outer join Habitats H
on     (A.hid = H.hid)
group by  H.name
having count(*) = 1
```

compare results of Q65 and Q66!

Q66

```
select H.name as habitat_name, count(*)
from   Animals A, Habitats H
where  A.hid = H.hid
group by  H.name
having count(*) = 1
```

More examples

Animals (aid, name, hid)

aid	name	hid
1	elephant	101
2	giraffe	101
3	whale	102
4	dolphin	102
5	okapi	null

Habitats (hid, name)

hid	name
101	African savanna
102	North Pacific ocean
103	Eurasian tundra

List names of habitats that are not known to be inhabited by any animals.

Q67

More examples

Animals (aid, name, hid)

aid	name	hid
1	elephant	101
2	giraffe	101
3	whale	102
4	dolphin	102
5	okapi	null

Habitats (hid, name)

hid	name
101	African savanna
102	North Pacific ocean
103	Eurasian tundra

List names of habitats that are not know to be inhabited by any animals.

Q67.a

```
select H.name as habitat_name
from   Habitats H left outer join Animals A
on     (H.hid = A.hid)
where  A.hid is null
```

Q67.b

```
select H.name as habitat_name
from   Animals A right outer join Habitats H
on     (H.hid = A.hid)
where  A.hid is null
```

More examples

Animals (aid, name, hid)

aid	name	hid
1	elephant	101
2	giraffe	101
3	whale	102
4	dolphin	102
5	okapi	null

Habitats (hid, name)

hid	name
101	African savanna
102	North Pacific ocean
103	Eurasian tundra

List names of habitats that are not know to be inhabited by any animals.

Q67.c

```
select H.name as habitat_name
from   Habitats H
where  H.hid in (
    select hid from Habitats
    EXCEPT
    select hid from Animals
)
```

```
select H.name as habitat_name
 from  Habitats H
EXCEPT
select H.name
 from Animals A, Habitats H
where A.hid = H.hid
```

Database modifications

Insertion

```
insert into R (A1, A2, ..., An) values (v1, v2, ..., vn)
```

Q68
```
insert into Animals (aid, name, hid)
values (6, 'zebra', 101)
```

Q69
```
insert into Animals (aid, name)
values (7, 'rhinoceros')
```

Deletion

```
delete from R where <condition>
```

Q70
```
delete from Animals
where name = 'zebra'
```

Q71
```
delete from Animals
where hid = 102
```

Q72
```
delete from Animals
where hid is null
```

Updates

```
update R set  <new value assignment> where <condition>
```

Q73
```
update Animals
set    hid = 101
where  hid is null
```

Q74
```
update Animals
set    name = upper(name),
       hid = 101
```

Views

A view is a table whose rows are not explicitly stored in the database but are computed as needed from a view definition.

- Recall that SQL is a closed (compositional) language: every query returns a relation that can be used as input to other queries
- That is, technically, every SQL query defines a view
- If we want to refer to the query (and its results) frequently - it's worthwhile to create a view so we can refer to that query by name

```
create view African_Animals (aid, name)
  as select A.aid, A.name
     from   Animals A, Habitats H
     where  A.hid = H.hid
     and    H.name like `%Africa%`
```

Views

```
create view African_Animals (aid, name)
  as select A.aid, A.name
     from   Animals A, Habitats H
     where  A.hid = H.hid
     and    H.name like '%Africa%'
```

```
select * from African_Animals
```

African_Animals (aid, name)

aid	name
1	elephant
2	giraffe

```
insert into Animals (aid, name, hid)
     values (6, 'gazelle', 101)
```

```
select * from African_Animals
```

African_Animals (aid, name)

aid	name
1	elephant
2	giraffe
6	gazelle

Habitats (hid, name)

hid	name
101	African savanna
102	North Pacific ocean
103	Eurasian tundra

Animals (aid, name, hid)

aid	name	hid
1	elephant	101
2	giraffe	101
3	whale	102
4	dolphin	102
5	okapi	
6	gazelle	101

Materialized views

```
create materialized view High_Utilizers (sid, num_res)
  as select   sid, count(*)
     from     Reserves
     group by sid
     having   count(*) >= 2
```

Reserves (sid, bid, day)

sid	bid	day
1	101	10/10/12
1	102	10/10/12
1	101	10/7/12
2	102	11/9/12
2	102	7/11/12
3	101	7/11/12
3	102	7/8/12
4	103	19/9/12

```
select * from High_Utilizers
```

sid	num_res
1	3
2	2
3	3

```
insert into Reserves (sid, hid, day)
     values (4, 102,'10-Oct-2020')
```

```
select * from High_Utilizers
```

sid	num_res
1	3
2	2
3	3

```
select     sid, count(*)
     from      Reserves
     group by  sid
     having    count(*) >= 2
```

sid	num_res
1	3
2	2
3	3
4	2