Additional operators: Do **not** add any power to the language, just make query specification more succinct.

Intersection (\( \cap \)): \( r_1 \cap r_2 \) = all tuples that are in \( r_1 \) and in \( r_2 \).

Say and students enrolled in 321 are 444:

\[ r_1 \cap r_2 \]

\[ r_1 - (r - s) \equiv \emptyset \equiv r \cap s \]

Theta Join (\( \bowtie \)): combines selection and cartesian product into one operation:

\[ \bowtie_{\theta} (r \times s) = r \bowtie s \bowtie s \] (relation scheme = \( R \bowtie s \))

Ex: name of all students enrolled in 321

\[ \pi \text{.name} (\text{students} \bowtie_{\theta} \text{enrolled}) \]

when \( \theta = (s.\text{sid} = e.\text{sid}) \land (e.\text{cnum} = 321) \)
The theta join is very general. Usually use more specific joins:

**Natural join (\( \text{\texttt{\textasteriskcentered}} \))**: Connect tuples from two relations that match on all common attributes. In the result, common attributes are kept only once.

**Example**: List of all students on the classes they are taking:

\[
\text{student} = (\text{name, sid, ssn, stats}) \quad \text{Enrolled} = (\text{sid, course})
\]

\[
\text{Scheme } (r) = (\text{name, sid, ssn, stats, course})
\]

Note, common attributes kept only once.

Say want just info on Sue:

\[
\sigma_{\text{name} = \text{Sue}} (\text{student} \text{\texttt{\textasteriskcentered}} \text{Enrolled})
\]

How express as a single \( \Theta \) join?

\[
(\text{student} \text{\texttt{\textasteriskcentered}} \text{Enrolled}) \quad \text{where } \Theta = (s.\text{sid} = e.\text{sid}) \lor (s.\text{name} = \text{Sue})
\]
Formal definition of \( \bowtie \):

\[
\text{let } R = \text{scheme of } r \\
S = \text{scheme of } s \\
A = R \cap S = \{\text{set of attributes in common}\} \\
R \bowtie S = R \times S \\
\text{where } \Theta = (r.A_1 = s.A_1) \wedge (r.A_2 = s.A_2) \wedge \ldots (r.A_n = s.A_n) \\
\forall A_i \in R \cap S
\]

we are usually only interested in equality of one attribute.

**Equi-join** \( (\bowtie_{\text{attr}}) \): Convert tuples from two relations that match on the specified attribute. Specified attribute kept; once, other common attributes denoted \( r_1.\text{attr} \cap r_2.\text{attr} \) in resultant scheme.

**Ex:** Find all names that are names of a student and a faculty member.

\( \text{TH} \) (student \( \bowtie \) name \( \bowtie \) faculty)
Q: Would \( \Pi_{\text{name}} (\text{student} \bowtie \text{faculty}) \) give the same answer?

<table>
<thead>
<tr>
<th>student</th>
<th>name</th>
<th>gid</th>
<th>ssn</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>22</td>
<td>936</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Sue</td>
<td>19</td>
<td>854</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Bob</td>
<td>17</td>
<td>716</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Ulf</td>
<td>49</td>
<td>815</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>faculty</th>
<th>name</th>
<th>fid</th>
<th>ssn</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom</td>
<td>18</td>
<td>321</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Sue</td>
<td>19</td>
<td>717</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Lisa</td>
<td>20</td>
<td>216</td>
<td>50</td>
<td></td>
</tr>
</tbody>
</table>

\[
\Pi_{\text{name}} (\text{student} \bowtie \text{faculty}) = \{\text{Sue}\}
\]

\[
\Pi_{\text{name}} (\text{student} \bowtie \text{faculty}) = \emptyset \quad \text{empty set}
\]

Why? Because \( \bowtie \) says all attributes in common must be equal and student Sue has a different ssn than faculty Sue.
Guide(s, sid, cum, semester, grade, fid)

Q: How set name and sid of all students who have taken a class by "lect"?

$\text{select } \text{faculty} \bowtie \text{Grades} \bowtie \text{student} \\
\text{on } \text{sid}$

Q: What is the scheme of $(\text{faculty} \bowtie \text{Grades} \bowtie \text{student})$?

$(\text{name, fid, f.ssn, salary, sid, cum, semester, grade, s.name, s.ssn, status})$

Must include name + f.ssn to distinguish from s.name + s.ssn.

Q: Would $\text{select } \text{name, sid} \\
\text{from } (\text{faculty} \bowtie \text{Grades} \bowtie \text{student})$ work?

No; would force equality at name + s.ssn also.

Q: Would $\text{select } \text{name, sid} \\
\text{from } (\text{Grades} \bowtie \text{faculty} \bowtie \text{student})$ work?

Yes, & are associative.
### Equi-join of 3 relational example

Faculty (name, fid, ssn, salary)

<table>
<thead>
<tr>
<th>fac</th>
<th>name</th>
<th>fid</th>
<th>ssn</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>lev</td>
<td>7</td>
<td>999</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>nicol</td>
<td>8</td>
<td>717</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>park</td>
<td>2</td>
<td>871</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

Student (name, sid, ssn, st.

<table>
<thead>
<tr>
<th>Student</th>
<th>name</th>
<th>sid</th>
<th>ssn</th>
<th>st.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>317</td>
<td>216</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Sue</td>
<td>389</td>
<td>285</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Harry</td>
<td>421</td>
<td>525</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Tom</td>
<td>516</td>
<td>219</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Grades (sid, cum, scnd, grade, cid)

<table>
<thead>
<tr>
<th>Grades</th>
<th>sid</th>
<th>cum</th>
<th>scnd</th>
<th>grade</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>317</td>
<td>100</td>
<td>92.1</td>
<td>B</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>317</td>
<td>101</td>
<td>92.1</td>
<td>B</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>421</td>
<td>201</td>
<td>92.1</td>
<td>A</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>516</td>
<td>321</td>
<td>92.2</td>
<td>A</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>317</td>
<td>321</td>
<td>92.2</td>
<td>B</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>389</td>
<td>301</td>
<td>92.1</td>
<td>A</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>217</td>
<td>444</td>
<td>91.3</td>
<td>A</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>389</td>
<td>444</td>
<td>91.3</td>
<td>B</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>421</td>
<td>444</td>
<td>91.3</td>
<td>C</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
$$\sigma_{\text{faculty}} \left[ \left( \sigma_{\text{fim} = \text{leut}} \left( (\sigma_{\text{sid} = \text{leut}}) \left( \sigma_{\text{Grades}} \right) \right) \right) \right] \left( \text{students} \right)$$

$$\sigma_{\text{fim} = \text{leut}} \left( (\sigma_{\text{sid} = \text{leut}}) \left( \sigma_{\text{Grades}} \right) \right) \left( \text{students} \right)$$

\[
\begin{array}{cccc}
\text{leut} & 7 & 999 & 10 \\
\text{"} & " & " & 516 & 221 \\
\text{"} & " & " & 317 & 321 \\
\text{leut} & 7 & 999 & 10 & 389 & 301
\end{array}
\]

\[
\begin{array}{cccc}
\text{leut} & 7 & 999 & 10 & 317 & 100 \\
\text{"} & " & " & 516 & 221 \\
\text{"} & " & " & 317 & 321 \\
\end{array}
\]

$$\text{\sigma_{\text{leut}} \left( \left( \text{students} \right) \right) = \left( \text{leut} \right) \text{\sigma_{\text{leut}}} \left( \left( \text{students} \right) \right) \left( \text{leut} \right)$$

\[
\begin{array}{cccc}
\text{fim} & \text{sid} & \text{Grades} & \text{students} \\
\text{leut} & 7 & 999 & 10 & 317 & 100 & \text{Bob} & 216 & 4 \\
\text{"} & " & " & 516 & 221 & \text{Tam} & 219 & 2 \\
\text{"} & " & " & 317 & 321 & \text{Bob} & 216 & 4 \\
\text{leut} & 7 & 999 & 10 & 389 & 301 & \text{Sue} & 285 & 4 \\
\end{array}
\]

$$\text{\sigma_{\text{leut}} \left( \left( \text{students} \right) \right) = \left( \text{leut} \right) \text{\sigma_{\text{leut}}} \left( \left( \text{students} \right) \right) \left( \text{leut} \right)$$

\[
\begin{array}{cccc}
\text{Bob} & 317 \\
\text{Tam} & 516 \\
\text{Sue} & 389 \\
\end{array}
\]
2) \( \sum \sum \sigma \text{faculty} \& \text{guides} \& \text{students} \)

\[
\text{faculty} \& \text{guides} =
\begin{array}{cccc}
\text{name} & \text{fid} & \text{ssn} & \text{salary} & \text{srnd} & \text{emr}
\hline
\text{lect} & 7 & 995 & 10 & 317 & 100
\text{lec} & 5 & 321 & 316 & 321
\text{lect} & 7 & 995 & 10 & 389 & 301
\text{nical} & 8 & 717 & 20 & 317 & 101
\text{park} & 2 & 871 & 30 & 421 & 201
\text{park} & 2 & 871 & 30 & 421 & 444
\end{array}
\]

\text{(count same guides for brevity)}

\text{faculty} \& \text{guides} \& \text{students} =
\begin{array}{cccc}
\text{faculty} & \text{guides} & \text{students}
\hline
\text{name} & \text{fid} & \text{sid}
\end{array}
\]

\text{Same as above with appropriate student attributes added on.}

\text{Now} \ \sum \ \sigma \text{faculty} \& \text{students} \text{eliminates all nical and park tuples.}

\text{Then} \ \sum \ \sigma \text{faculty} \& \text{students} \text{gives students who have taken a class taught by lect.}
What if I want most of all students who took a class by "lect" and have gotten an A in at least one of the classes?

How about:

\[ \text{grade} = "A" \quad \forall \text{s.name, sid} \quad \sigma \quad [ \text{faculty} \land \text{grade} \land \text{student} ] \]

No: after the projection grade info has been lost

\[ \text{grade} = "A" \text{ must be done before } \forall \text{s.name, sid} \]

Ex:

\[ \forall \text{s.name, sid} \quad \sigma \quad [ \text{faculty} \land \text{grade} \land \text{student} ] \quad (\text{finam = lect}) \land (\text{grade} = "A") \]
Division (\( \div \))

Ex: Find students who completed all courses offered in the past.

\[
S = \prod_{\text{cum \ (grades)}} \\
\]

\[
r = \prod_{\text{sid, cum \ (grades)}} \\
\]

\[
\text{answer} = r \div S \\
\text{scheme} (r \div S) = R - S = 3 \text{sid}^2 \\
\]

All students in \( r \) that have taken all courses in \( S \). (Built on basic operation)

Ideas: find all \( x \) values that are disqualified.

\[
disqualified = \prod_{\text{sid}} \left[ \prod_{\text{sid}} (r) \times S \right] - R \\
\]

\[
r \div S = \prod_{\text{sid}} (r) - \prod_{\text{sid}} \left[ \prod_{\text{sid}} (r) \times S \right] - R \\
\]
Example

$S = \pi_{\text{cum}}(\text{grades}) = \{100, 101, 200, 301\}$

$r = \pi_{\text{sid, cum}}(\text{grades}) =$

<table>
<thead>
<tr>
<th>sid</th>
<th>cum</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>100</td>
</tr>
<tr>
<td>18</td>
<td>100</td>
</tr>
<tr>
<td>17</td>
<td>101</td>
</tr>
<tr>
<td>18</td>
<td>200</td>
</tr>
<tr>
<td>18</td>
<td>200</td>
</tr>
<tr>
<td>19</td>
<td>200</td>
</tr>
<tr>
<td>17</td>
<td>301</td>
</tr>
<tr>
<td>19</td>
<td>301</td>
</tr>
</tbody>
</table>

Is "18" $\in$ r\-s?  No

Is "19" $\in$ r\-s?  No

Is "17" $\in$ r\-s?  Yes

* Can show. $R \div S$ is composed of 5 fundamental operators.
Another example

workson (essn, pno, hours)
employee (name, essn, bdate, address, sex, ssn (dir))

Say want name of all employees who work on all the projects that "poe" works on.

1) \[ pno - puc \leftarrow \prod_{pno} \text{workson} \times \sigma_{essn=poe} (\text{employee}) \]
2) create relation of \[ \langle pno, essn \rangle \]

\[ \text{temp} \leftarrow \prod_{pno, essn} (\text{workson}) \]
3) Do division

\[ \text{temp2} \leftarrow (\text{temp} : \frac{\text{poe} - puc}) \]

Now have essn of all people who work on every project that smith does, set name

4) result \[ \leftarrow \Pi_{name} (\text{temp2} \times \text{employee}) \]
Transforming E/R → Relational Design

Entity Sets => Relations

Movies
- title
- year
- length
- filetype

Stars
- name
- address

Studios
- name
- address

Relationships

1) for each entity set involved we take its key attributes
2) add any relationship attributes
StarsIn (title, year, name, address, primaryStar)

OwNs (title, year, name)

* Def should also specify foreign keys used in Oracle (SQL)

Create table StarsIn (title varchar (20),
year number,
name varchar (20),
address varchar (20),
primaryStar char (3),
PRIMARY KEY (title, year, name, address),
FOREIGN KEY (title, year) REFERENCES movies,
FOREIGN KEY (name, address) REFERENCES Stars);
Relationships that are N : 1, 1 : N, or 1 : 1 can be optimized.

Example

Create table department (did INTEGER,
  varchar varchar (40),
  budget real,
  ssn char (11),
  PRIMARY key (did),
  foreign key (ssn) references employee);

If relationship key exists (ex. "since") also add it to the department relationship.

In general, if 1 : N just add the key from the "1" side into the relation entity set on the "N" side.
Further Examples

Sailors (sid, name, rating)
Boats (bid, buame, color)
Reserve (sid, bid, date)

1) Find name of sailors that have reserved boat #2

\[ \Pi_{\text{name}} (\sigma_{\text{bid} = 2} (\text{Reserve} \bowtie \text{Sailors})) \]

\[ \Pi_{\text{sid}} (\text{Reserve} \bowtie \text{Sailors}) \]
where \( \Theta = (r, \text{sid} = s, \text{sid}) \land (r, \text{bid} = 2) \)

2) Find name of sailors who have reserved a red boat.

\[ \Pi_{\text{sid}} (\text{Sailors} \bowtie (\text{Reserve} \bowtie (\sigma_{\text{color} = \text{red}} (\text{Boats})))) \]

3) Find colors of boats reserved by Pit

\[ \Pi_{\text{color}} [ (\sigma_{\text{smw = Pit}} (\text{Sailors}) \bowtie \text{Reserve} \bowtie \text{Boats})] \]

where can I use the \( \sigma \) to?
4) Find names of sailors who have reserved at least one boat:

\[ \prod_{\text{sid}} (\text{Reserve} \times \text{Sailor}) \]

5) Find the names of the sailors who have reserved all boats:

\[ \prod_{\text{sid}} \left( \left( \frac{\prod_{\text{bid}} (\text{Reserve})}{\prod_{\text{bid}} (\text{boats})} \right) \times \text{Sailors} \right) \]

6) Find names of sailors who have reserved both a red and green boat:

\[ \prod_{\text{sid}} \left( \left( \text{Sailors} \times \text{Reserve} \times \left( \prod_{\text{bid}} \text{colored} \right) \right) \right) \cap

\left( \text{Sailors} \times \text{Reserve} \times \left( \prod_{\text{bid}} \text{colored} \right) \right) \]

7) Find names of sailors who have reserved all red boats:

Same as (5) except boats replaced with

\[ \left( \prod_{\text{colored}} \left( \text{boats} \right) \right) \]
8) Find names of students who have received a red or green belt.

See as (6) except \( U \) instead of \( V \)

Disadvantages of Relational Algebra:

It is a procedural language, i.e. user must specify how system will get data.

Solution: Relation Calculus Languages

1) tuple relational calculus
2) QUEL
3) SQL
SQL

SQL: Structured Query Language (also known as SeQuel)

general form: \[
\begin{array}{l}
\text{Select } [\text{Distinct}] \text{ Select-list} \\
\quad \text{From} \text{ from-list} \\
\quad \text{where} \text{ qualification}
\end{array}
\]

\[
\begin{aligned}
\text{Select } A_1, A_2, \ldots, A_n \\
\quad \text{From } r_1, r_2, \ldots, r_m \\
\quad \text{where } p
\end{aligned}
\]

select \equiv \text{project }!! \text{ Bad name}

\[
\equiv \Pi_{A_1, A_2, \ldots, A_n} (\sigma_p (r_1 \times r_2 \times \ldots \times r_m))
\]

Scalors (sid, sname, unit)
Bots (bid, bname, color)
Rescue (sid, bid, date)
Sailors (sid, surname, rating)
boats (bid, bname, color)
reserve (sid, bid, date)

1) Find the base of all Sailors

\[ \pi_{\text{surname}}(\text{Sailors}) \Rightarrow \text{Select surname from Sailors} \Rightarrow \text{Select S.surname from Sailors S} \]

2) Find unique names of sailors

\[ \text{Select distinct S.surname from Sailors S} \]
*Note: without `distinct`, you get duplicates (``multiset``)

3) Find names of sailors whose rating is > 5

\[ \text{select S.surname from Sailors S where S.rating > 5} \]

4) Find name of sailors who have reserved boat #2

\[ \text{Select surname from Sailors, reserve where (sailors, sid = reserve.sid) and (bid = 2) +} \]

Comments:
- Can put `reserve.bid` but not needed since unambiguous.
- Best to be explicit +
- Sid, `reserrove.bid` or
- Use `table version`
SELECT S.s_name, S.v_name + 1 AS rating
FROM Select S, Return R1,
WHERE S.s_id = R.s_id AND R.r_id = "17"

* This query returns the each score for each side,
** Select 12 rows from Return 1 **

⇒ Each side's qualification can be generalized that:

expv1 = expv2

SELECT S1.s_name AS name1, S2.s_name AS name2
FROM Select S1, Select S2
WHERE 2 * S1.rating = S2.rating

* Reports scores of each name for whom 1st
side's rating is 2x the second side's rating.

* Note: the tuple variables may over Selects
Merging of an SQL query:

1) compute \( X \) from the FROM-list
2) delete tuples that fail qualification condition
3) delete all columns not in the SELECT-list
4) if DISTINCT specified, remove duplicates

4.5) Find the name of sailors who have rescued at least one boat:

Select `S.Name`
From `Sailor S`, `Rescue R`
Where `S.sid = R.sid`

Expressions & Strings in the SELECT Clause

SQL supports more general version of column list:

Expression AS column-name

>`\Rightarrow` remove old or create new

任意, feeding qualitative or story expression, column has
Find union of sailors who have reserved at least the different boats for some day.

Select S, S1, S2, R1, R2
From S, S1, S2, R1, R2

where (S.sid = R1.sid) AND (S.sid = R2.sid)
AND (R1.date = R2.date)
AND (R1.bid < R2.bid)

leave off and ask if okay

String Pattern Matching: LIKE operator

% = zero or more arbitrary char
_ = exactly one arbitrary char

Ex:
- A0% $\Rightarrow$ any 3 or more char start with 2nd and 3rd char are A0

Find all sailors whose name begin as "Scott"

Select *
From S

where S.name LIKE 'Scott%'

- Find the name of sailors we have rescued a red or a green boat

```
SELECT S.name
FROM Sailors S, Bids B, Rescue R
WHERE (S.sid = R.sid) AND (R.bid = B.bid) AND (B.color = 'red' OR B.color = 'green')
```

- Find the name of sailors we have rescued a red and a green boat

\[ \cup \]

```
SELECT S.name
FROM Sailors S, Bids B1, Rescue R1, Bids B2, Rescue R2
```

UCLI: By using set operators this can be easier to express
Firstly, the "or" query:

Select S.name
From SAILORS S, DOCKS D, RESERVE R
where S.sid = R.sid and R.bid = D.bid and D.color = 'red'
UNION
Select S2.name
From SAILORS S2, DOCKS D2, RESERVE R2
where S2.sid = R2.sid and R2.bid = D2.bid and D2.color = 'green'

Now, how set the "red and green"?

⇒ just replace UNION with INTERSECT

How set names of sailors who have reserved red boats but not green boats?

⇒ replace UNION with EXCEPT

* Union, Intersect, Except can be used on any two tables that are "join-compatible"—i.e., have the same # of columns & the columns taken in order have the same domains.
Find all sids of sailors who have a Rating of 10 or have reserved lot #111.

```
Select S.sid
From Sailors S
Where S.rating = 10
Union
Select R.sid
From Reserve R
Where R.bid = 111
```

* Unlike all other SQL commands, the default is to **eliminate duplicates**. If you want duplicates:

```
Union All
```
Nested Queries:

After you want to express a condition that is itself the result of a query.

Example 1:

Find the names of sailors who have reserved boat #11.

```
SELECT S.name
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                 FROM Reserve R
                 WHERE R.bid = 11)
```

- Easy to modify to find sailors who have not reserved #11.
  Just replace `IN` with `NOT IN`.

Example 2:

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

```
SELECT S.name
FROM Sailors S
WHERE S.sid IN (SELECT R.sid
                 FROM Reserve R
                 WHERE R.bid IN (SELECT B.bid
                                  FROM Boats B
                                  WHERE B.color = 'red'))
```
1) What if I replaced outer \( \text{IN} \Rightarrow \text{NOT IN} \)?
   - Get names of sailors who have not rescued a red boat.

2) What if replace inner \( \text{IN} \Rightarrow \text{NOT IN} \)?
   - Get names of sailors who have rescued a boat but the boat was not red.

3) What if replace both \( \text{IN} \Rightarrow \text{NOT IN} \)?
   - Names of sailors who have, but rescued a boat that is not red.
   - Does it mean they rescued a red boat? \( \Rightarrow \text{NO} \)

Correlated Nested Queries:

Find names of sailors who have rescued boat #100:

Select S. name
From Sailors S
Where EXISTS (SELECT *
From Rescue R
Where R.bid = 100
And (S.sid = R.sid))

\( \forall \) tuple in S, check to see if set of Rescue has R
s.t. R.bid = 100 + S.sid = R.sid is not empty.
\[ \exists \text{ Vebice} \quad \text{EXISTS} \quad \equiv \quad \text{with NOT EXISTS} \]

\[ \equiv \text{max of Sailer who have NOT rescue in #100} \]

**UNIQUE** : return true if no duplicates

**NOT UNIQUE** : true if duplicates

```sql
SELECT S.ssid
FROM Section S
WHERE NOT UNIQUE (SELECT * FROM Rescue R
WHERE R.ssid = 100
AND S.ssid = R.ssid)
```

\[ \Rightarrow \text{return max of Sailer who have rescue boat #100 at least twice} \]

**Set Operators**

- `EXISTS`, `IN`, `UNIQUE`,
- `op ANY`, `op ALL`
- `op \in \{ <, <=, =, >, >=, > \}`

Find Sailer whose vote is better than \( \forall \) \\
\[ \text{Select}\ s.ssid\]
\[ \text{From}\ Section\ S\]
\[ \text{where } S.voting > \text{ANY} \]

```sql
SELECT S2.voting
FROM Section S2
WHERE S2.sname = 'Vebice'
```

- Is this correlated?
  - **No!**
Note: returns Sids of sailors whose rating is greater than some sailor's rating.

If there are multiple ratings, and you want a sailor whose rating is > all ratings returned:

SELECT => replace ANY with ALL

What if no sailors exceed rating?

a) > ANY => return false
   (since none found a case when rating > least)

b) > ALL => return true
   because rating = least for all sailors

Find the sailor with the highest rating:

SELECT S.sid
FROM Sailor S
WHERE S.rating >= ALL (SELECT S2.rating
                        FROM Sailor S2)

Note: IN = = ANY
     NOT IN = < > ALL
New Notice Queries

Find names of sailors who have reserved a red and green boat.

```
Select S.name
From Sailor S, Boat B, Reserve R
where S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
  AND S.sid IN (Select S2.sid
  From Sailor S2, Boat B2, Reserve R2
  where S2.sid = R2.sid AND R2.bid = B2.bid
  AND B2.color = 'green')
```

All sailors who have reserved a red boat and
have sailer ids of sailors who have reserved a given boat.

This query will only interest can be used in

How to do division?

Find the list of sailors who have reserved all boats.

```
Select S.name
From Sailor S
where NOT EXISTS ((Select R.bid
  From Boat B
  WHERE B.id = R.bid)
  EXCEPT
  (Select R.bid
  From Reserve R
  WHERE R.sid = S.sid))
```
Ordering display of results

```sql
select name, sid
from student
where status = 3
order by name
```

The above query lists all junior in alphabetical order.
If we reverse alphabetical order:
```
order by name desc
```

desc $\equiv$ descending
gsc $\equiv$ gscending

May have duplicate names, if want secondary sort value:

```sql
select name, sid
from student
where status = 3
order by name asc, sid desc
```
SQL Aggregates
- `avg`, `max`, `sum`, `count`

1) Select `sailor` (rating), `avg` (rating), `max` (rating) from `sailors`

2) Find average rating of sailors who have been sailing less than 2 years:

   Select `avg` (rating)
   from `sailors`
   where `yrs-sailed` < 2

3) Retrieve the number of sailors who have sailed for more than 5 years:

   Select `count` (*)
   from `sailors`
   where `yrs-sailed` > 5

4) Retrieve the number of sailors who have reserved boat #161 on 01/31/95:

   Select `count` (*)
   from `reserve`
   where `bid` = 161 and `date` = 01/31/95
5) Find the average age of sailors with age > 10

```
SELECT AVG(s.age) 
FROM sailors s 
WHERE s.age > 10
```

6) Find the name + age of the oldest sailor:

```
SELECT s.name, MAX(s.age) 
FROM sailors s
```

*Illegal*

If the `SELECT` clause uses an aggregate, the `FROM` clause can only use aggregate or arbitrary function "group by".

```
SELECT s.name, s.age 
FROM sailors s 
WHERE s.age = (SELECT MAX(s.age) 
FROM sailors s)
```
\[
\text{Select } s, s_{\text{sum}} \\
\text{from } S\text{alers } S \\
\text{where } s.\text{rating} > \text{Avg}(s.\text{rating}) \\
\]

\[
\text{NC} \\
\Rightarrow \text{Select } s, s_{\text{sum}} \\
\text{from } S\text{alers } S \\
\text{where } s.\text{rating} > (\text{Select } \text{Avg}(s'.\text{rating}) \\
\text{from } S\text{alers })
\]
7) Retrieve the name of sailors whose rating is above the average rating (over all sailors)
   select name
   from sailors
   where rating > AVG(rating)

Groupings attributes: when we want to apply
the aggregates to subgroups of tuples.

8) Retrieve the age and average rating
    for sailors of each age

   select age, AVG(rating)
   from sailors
   group by age

Ex.

sailors

<table>
<thead>
<tr>
<th>age</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>28</td>
</tr>
<tr>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
</tr>
<tr>
<td>4</td>
<td>28</td>
</tr>
<tr>
<td>1</td>
<td>24</td>
</tr>
</tbody>
</table>

15, 5
21, 4
24, 2
28, 5
9) Find the age of the youngest voter who is eligible to vote for each entry level and at least 2 seconds:

```
SELECT S.vote_yr, MIN(s.vote_age)
FROM Voters S
WHERE S.vote_age > 18
GROUP BY S.vote_yr
HAVING COUNT(*) > 1
```

See 9.70 for example of how processor

10) For each not bid, find the # resources:

```
SELECT B.bid, COUNT(*) AS resource_count
FROM Bids B, Bidders B, Reserve R
WHERE B.bid = R.bid AND R.bid = B.bid AND B.attri = 'not'
GROUP BY B.bid
```
Assume following sailors instance:

<table>
<thead>
<tr>
<th>Sid</th>
<th>Sname</th>
<th>Rating</th>
<th>95C</th>
</tr>
</thead>
<tbody>
<tr>
<td>85</td>
<td>rvt</td>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>31</td>
<td>rubber</td>
<td>8</td>
<td>55.5</td>
</tr>
<tr>
<td>32</td>
<td>andy</td>
<td>8</td>
<td>25.5</td>
</tr>
<tr>
<td>95</td>
<td>bob</td>
<td>3</td>
<td>63.5</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
<td>10</td>
<td>35.0</td>
</tr>
<tr>
<td>29</td>
<td>ting</td>
<td>1</td>
<td>33.0</td>
</tr>
<tr>
<td>71</td>
<td>jessie</td>
<td>10</td>
<td>16.0</td>
</tr>
</tbody>
</table>

Step 4: Create cross product

⇒ Since each value ⇒ sailors

Step 2: Apply where clause

⇒ eliminates jessie

Step 3: Eliminate unneeded columns

Needed: anything in select, group by and order by clause

⇒ this eliminates sid, sname

Step 4: sort relation according to step by clause

```
| 1   | 33 |
| 3   | 25.5 |
| 3   | 63.5 |
| 8   | 55.5 |
| 8   | 25.5 |
| 10  | 16.0 |
```
Step 5: apply given qualification

\[ \text{evidence} \text{ raw } \text{ with } 1 \text{ to } 10 \]

*Note order of where + having important

Step 6: apply the associate & select to

- varying groups

\[
\begin{array}{l}
3 \quad 25.5 \\
8 \quad 25.5
\end{array}
\]

If going to apply
because what jessie
would be her mother
and would be her,
answer to noting of
(3, 8 + 10) em (4)

See an NOT 2 or are 18 years old
in the graph of "hesi"
1) 
```
SELECT S bid, COUNT(*) AS subcount
FROM Sales, Parts P, Reserve R
WHERE (R bid = S bid) AND (P code = "red")
GROUP BY S bid
HAVING 0.0 <= subcount
```

Illegal

Can do by sum
where (R bid = S bid) and (P code = "red")

* only columns that appear in group by clause can be in having clause unless they appear as arguments to an aggregate

2) Find the avg size of sailors for each rating level that has at least two sailors.

```
SELECT S rating, AVG(S.size) AS avg_size
FROM SAILORS S
GROUP BY S rating
HAVING 1 <= (SELECT COUNT(*)
            FROM SAILORS S2
            WHERE S rating = S2.rating)
```

or

```
HAVING 1 <= COUNT(*)
```
Movie (title, year, length, in Color, studioName, producerC#)
MovieExec (name, address, cert#, network)

1) Find the average net worth of all movie execs:
   Select AVG(networth)
   From MovieExec

2) Find the total net worth of movie execs for each studio (total by net worth for each movie)
   Select studioName, SUM(networth)
   From Movie
   Group by studioName;

3) Find the total length of films produced by each producer listed by name
   Select name, SUM(lengths)
   From MovieExec
   Having sum(lengths) > 0
   Where M.producerC# = NE.cert#
   Group by NE.name
What if two different producers have same name?

```
SELECT NE.name, NE.coi#, SUB(1000,)
FROM MovieExec NE, Movie M
WHERE M.producer CH = NE.coi#
GROUP BY NE.name, NE.coi#
```

4) Assume producer names are unique. We want to print total film legs for these producers who have both at least one film before 1970.

```
SELECT NE.name, SUB(1000,)
FROM MovieExec NE, Movie M
WHERE M.producer CH = NE.coi#
GROUP BY NE.name
HAVING MIN(yr, M.year) < 1970
```
**Data Definition Languages**

**SGL**
- *Set scheme for each relation*
- *Domain of attribute values*
  - Which indices exist on each relation
  - Security and authorization info for each relation
  - Integrity constraints
  - Physical storage structure of each relation

Consider only *N* items now

**Define relation scheme (including domain of attributes):**

```
create table R (A_1 D_1, A_2 D_2, ..., A_n D_n)
```

- `create` creates a new relation (empty)
- `use` insert to add tuples

```
drop table R
```

Drop table removes all data in R and all info about R

**QUEL:** similar commands exist for QUEL
QBE: Query By Example

QBE System developed at IBM T.J. Watson, "user friendly" DBL.

Queries expressed as examples:

System provide "skeleton title" which user modifies to show what is wanted.

- Print the field:
  \[ P \]

- Variable name:
  \[ X \]

<table>
<thead>
<tr>
<th>student</th>
<th>name</th>
<th>sid</th>
<th>ssn</th>
<th>stat</th>
</tr>
</thead>
</table>

System supplies this skeleton:

<table>
<thead>
<tr>
<th>student</th>
<th>name</th>
<th>sid</th>
<th>ssn</th>
<th>stat</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ P ]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

⇒ Prints out name of all students.

⇒ QBE eliminates duplicates automatically, if want duplicates include must specify as:

(P, 91)
### Print names of all juniors

<table>
<thead>
<tr>
<th>students</th>
<th>name</th>
<th>sid</th>
<th>s54</th>
<th>stat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Vocabular used in joins:

**What does this do?**

<table>
<thead>
<tr>
<th>students</th>
<th>name</th>
<th>sid</th>
<th>s54</th>
<th>stat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>grades</th>
<th>sid</th>
<th>cum</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>321</td>
<td></td>
</tr>
</tbody>
</table>

Answer: prints out names of all sophomores enrolled in 321