Division: / operator in Relational Algebra

- Division is used to answer queries such as:
  
  *Find sailors who have reserved all boats.* (or all blue boats)
  
  *Find suppliers that catalog all green parts.*

- Let $A$ have 2 fields, $x$ and $y$; $B$ have only field $y$:
  
  $$A/B = \left\{ \langle x \rangle \mid \exists \langle x, y \rangle \in A \quad \forall \langle y \rangle \in B \right\}$$

- $A/B$ contains all $x$ tuples (sailors) such that for every $y$ tuple (boat) in $B$, there is an $xy$ tuple in $A$ (reservation by $x$ for $y$)

- Or, if the set of $y$ values (boats) associated with an $x$ value (sailor) in $A$ contains all $y$ values in $B$, the $x$ value is in $A/B$.

- In general, $x$ and $y$ can be any sets of fields (not just singletons, but we won't cover those cases)
Note that Sailor sid2 reserved all boats

Let A = sid and bid columns of reserves (as table)
Let B = bid column of boats (as table)
Then A/B = table containing only sid sid2, the answer to the query about which sailors reserved all boats.
If $A = B \times C$, it’s simple...

- If $A = B \times C$, then $A/B = C$ and $A/C = B$
- This means $A$ has all possible $(x, y)$ pairs
- Example: $A =\begin{array}{|c|c|}
\hline
\text{sid} & \text{bid} \\
\hline
s1 & b1 \\
\hline
s1 & b2 \\
\hline
s1 & b3 \\
\hline
s2 & b1 \\
\hline
s2 & b2 \\
\hline
s2 & b3 \\
\hline
\end{array}$ $B =\begin{array}{|c|}
\hline
\text{sid} \\
\hline
s1 \\
\hline
s2 \\
\hline
\end{array}$ $C =\begin{array}{|c|}
\hline
\text{bid} \\
\hline
b1 \\
\hline
b2 \\
\hline
b3 \\
\hline
\end{array}$
But most \((x,y)\) relationships are more complicated...

Example: \(A = \text{sid and bid columns of reserves} =\)

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>b1</td>
</tr>
<tr>
<td>s1</td>
<td>b2</td>
</tr>
<tr>
<td>s1</td>
<td>b3</td>
</tr>
<tr>
<td>s1</td>
<td>b4</td>
</tr>
<tr>
<td>s2</td>
<td>b1</td>
</tr>
<tr>
<td>s2</td>
<td>b2</td>
</tr>
<tr>
<td>s3</td>
<td>b2</td>
</tr>
<tr>
<td>s4</td>
<td>b2</td>
</tr>
<tr>
<td>s4</td>
<td>b4</td>
</tr>
</tbody>
</table>

- Here, sailor s1 reserved all boats (b1-b4), but the others reserved an incomplete set of boats.
- So there’s no “factoring” of \(A\) into two simple tables, but we still want to find the sailors who reserved all boats, or a certain set of boats.
Examples of Division A/B

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>b1</td>
</tr>
<tr>
<td>s1</td>
<td>b2</td>
</tr>
<tr>
<td>s1</td>
<td>b3</td>
</tr>
<tr>
<td>s1</td>
<td>b4</td>
</tr>
<tr>
<td>s2</td>
<td>b1</td>
</tr>
<tr>
<td>s2</td>
<td>b2</td>
</tr>
<tr>
<td>s3</td>
<td>b2</td>
</tr>
<tr>
<td>s4</td>
<td>b2</td>
</tr>
<tr>
<td>s4</td>
<td>b4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
</tr>
<tr>
<td>b2</td>
</tr>
<tr>
<td>b3</td>
</tr>
<tr>
<td>b4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sid</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
</tr>
</tbody>
</table>

A/B

- A/B finds the x values (si) in A that have all the y values (bi) as <x,y> rows in A.
- Here the sailors who reserved all four boats in B.
Examples of Division $A/B$ with smaller B-sets, i.e. smaller “for all” sets

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>bid</th>
<th>bid</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>b1</td>
<td>b2</td>
<td>b2</td>
<td>b1</td>
</tr>
<tr>
<td>s1</td>
<td>b2</td>
<td></td>
<td>b2</td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>b3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s1</td>
<td>b4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>b1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>b2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>b2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>b2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>b2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>b4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$A/B1$, $A/B2$, $A/B3$
Division problems we will cover

• Although it is possible to use division with various sized tables, we will only cover cases of A / B for 2-column A and 1-column B.
• This will cover the common case of A being the “relationship” table ( pared down to the two id columns) and B being the id column of one of the related entities, the “for all” condition.
• For example, the table of sid, bid of the last slide is

\[ A = \pi_{\text{sid,bid}} \text{Reserves} \]

and table of bids is
\[ B = \pi_{\text{bid}} \text{Boats} \]
• So the division is

\[ (\pi_{\text{sid,bid}} \text{Reserves})/(\pi_{\text{bid}} \text{Boats}) \]
• Providing the set of sids of sailors who reserved all the boats.
Query 1

“Find the names of sailors who’ve reserved all boats”

$$\rho(\text{Tempsids}, (\pi_{\text{sid}, \text{bid}} \text{Reserves})/ (\pi_{\text{bid}} \text{Boats}))$$

$$\pi_{\text{sname}} (\text{Tempsids} \bowtie \text{Sailors})$$
Query 2

“Find sailors who’ve reserved all red boats”

$$\rho(\text{Temp}, (\pi_{\text{sid, bid}} \text{Reserves}) / (\pi_{\text{bid}} (\sigma_{\text{color='red'}} \text{Boats})))$$

$$\pi_{\text{sname}}(\text{Temp} \bowtie \text{Sailors})$$
Expressing A/B Using Basic Operators

• For A/B, compute all x values that are not **disqualified** by some y value in B
  • x value is **disqualified** if by attaching y value from B, we obtain an xy tuple that is not in A
    ▪ All possible xy tuples: \( \pi_x(A) \times B \)
    ▪ xy tuples not in A: \( \pi_x(A) \times B \) – A

**Disqualified x values:** \( \pi_x((\pi_x(A) \times B) – A) \)

**A/B:** \( \pi_x(A) – \) all disqualified tuples

\[ \pi_x(A) – \pi_x((\pi_x(A) \times B) – A) \]

Note that this is impressive, but not useful in practice. It does show one (inefficient) way to do division in SQL.
Find disqualified x’s for A/B

Example: \( A = \)

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>b1</td>
</tr>
<tr>
<td>s1</td>
<td>b2</td>
</tr>
<tr>
<td>s1</td>
<td>b3</td>
</tr>
<tr>
<td>s1</td>
<td>b4</td>
</tr>
<tr>
<td>s2</td>
<td>b1</td>
</tr>
<tr>
<td>s2</td>
<td>b2</td>
</tr>
<tr>
<td>s3</td>
<td>b2</td>
</tr>
<tr>
<td>s4</td>
<td>b2</td>
</tr>
<tr>
<td>s4</td>
<td>b4</td>
</tr>
</tbody>
</table>

- Here, sailor s1 reserved all boats \((b1-b4)\), but the others reserved an incomplete set of boats
- Example: sailor s2 doesn’t have \(<s2, b3>\), so \(x = s2\) is disqualified from \(A/B\).

\[
\pi_x(A) = (s1, s2, s3, s4)
\]

\[
\pi_x(A \times B) = ((s1, b1), \ldots (s4, b4)) \quad (16 \text{ tuples})
\]

\[
(\pi_x(A \times B) - A) = ((s2, b3), (s2, b4), \ldots)
\]

\[
\pi_x((\pi_x(A \times B) - A)) = (s2, s3, s4)
\]

= all disqualified tuples, leaving only s1 qualified
Division in SQL

- Not supported as primitive operator

- Need to use nested queries to express division
  - One of the most subtle queries in SQL
  - Need to pay close attention to writing SQL division queries!

- There are two well-known ways of writing division queries
  - Using the set `EXCEPT` operator (2-level nesting)
  - Without the `EXCEPT` operator (3-level nesting)
"Find sailors who’ve reserved all boats."

With **EXCEPT** (use **MINUS** in Oracle):

```
SELECT  S.sname
FROM    Sailors S
WHERE   NOT EXISTS
    ( -- look for bid(s) unconnected via R to this sid
      (SELECT  B.bid  FROM  Boats B)
     EXCEPT -- ones connected via R to S.sid
       (SELECT  R.bid  FROM  Reserves R
        WHERE  R.sid=S.sid)
    )
```

Note: the parentheses around the two inner SELECTs are not needed:
“subquery EXCEPT subquery” qualifies as a subquery, and the two are surrounded by parentheses needed by exists (...
Division: Solution 1 (Q9, pg. 150)

“Find sailors who’ve reserved all boats.”

With **EXCEPT** (use **MINUS** in Oracle):

```
SELECT  S.sname
FROM    Sailors S
WHERE   NOT EXISTS
        ( -- look for bid(s) unconnected via R to this sid
            (SELECT  B.bid FROM  Boats B)
        EXCEPT    -- ones connected via R to S.sid
            (SELECT  R.bid FROM  Reserves R
             WHERE  R.sid=S.sid)
        )
```

• Note that this is not just Reserves/Boats, but actually
  Reserves/Boats (finding the sids) joined with Sailors (to get the snames)
• It is possible to rewrite it to be purely Reserves/Boats (and get the sids)
• We just use Reserves instead of Sailors to provide the outer loop over sids…
Division: Solution 1, using EXCEPT

“Find sailor sids who’ve reserved all boats”
using just Reserves/Boats

```
SELECT  DISTINCT R1.sid FROM  Reserves R1
 WHERE  NOT EXISTS
 (   -- look for bid(s) unconnected via R to this sid
 (SELECT  B.bid FROM  Boats B)
 EXCEPT  -- ones connected via R to R1.sid
 (SELECT  R.bid FROM  Reserves R
 WHERE  R.sid=R1.sid)
 )
```
Division: Solution 2 (also on pg. 150)

“Find sailors who’ve reserved all boats.”

Without **EXCEPT**: so works on mysql

```sql
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (SELECT B.bid
FROM Boats B
WHERE NOT EXISTS (SELECT *
FROM Reserves R
WHERE R.bid=B.bid
AND R.sid=S.sid))
```

Sailors S such that there is no boat B ...

without a Reserves tuple showing S reserved B
Division: Solution 3 (not in text)

“Find sailors who’ve reserved all boats.”

```
SELECT r.sid
FROM Reserves r
WHERE r.bid IN (SELECT b.bid from Boats b)
GROUP BY r.sid
HAVING COUNT(distinct bid) = (SELECT COUNT(*) FROM Boats)
```

The HAVING group for a certain sid qualifies the sid if it contains all the bids, shown by proper count. The distinct is needed because there can be multiple Reserves rows for one sid-bid combination (R is not a binary relationship table)

Note: if Reserves can have null sids, this solution can report a null as an output, unlike the earlier solutions, because GROUP BY groups nulls together as a legitimate group. (But we will only use division on non-null values.)
“Find sailors who’ve reserved all red boats.”

“Find sailors who’ve reserved all red boats.”

With \texttt{EXCEPT}:

\texttt{SELECT S.sname}
\texttt{FROM Sailors S}
\texttt{WHERE NOT EXISTS (}
\texttt{(SELECT B.bid FROM Boats B}
\texttt{WHERE B.color = ‘red’)}
\texttt{EXCEPT}
\texttt{(SELECT R.bid FROM Reserves R}
\texttt{WHERE R.sid=S.sid) )}
“Find sailors who’ve reserved all red boats.”

“Find sailors who’ve reserved all red boats.”

Without **EXCEPT**: Uses only Entry SQL-92, so portable

```sql
SELECT  S.sname
FROM   Sailors S
WHERE  NOT EXISTS (SELECT  B.bid
                    FROM   Boats B
                    WHERE  B.color='red' AND
                            NOT EXISTS (SELECT  *
                                         FROM   Reserves R
                                         WHERE  R.bid=B.bid
                                               AND R.sid=S.sid))
```
Another Example

Movies (movie_id, title, year, studio)
Actors (actor_id, name, nationality)
StarsIn(actor_id, movie_id, character)

“Find names of actors who star in ALL movies produced by Universal in year 1990.”

SELECT ???
Another Example

Movies (movie_id, title, year, studio)
Actors (actor_id, name, nationality)
StarsIn (actor_id, movie_id, character)

“Find names of actors who star in ALL movies produced by Universal in year 1990.”

SELECT A.name FROM Actors A
WHERE NOT EXISTS(

SELECT M.movie_id FROM Movies M
WHERE M.year=1990 AND M.studio='Universal'

EXCEPT

SELECT S.movie_id FROM Stars_In S
WHERE S.actor_id=A.actor_id
)


“Find sids and names of suppliers who sell all the green parts.”

```
SELECT s.sid, s.sname FROM suppliers s
WHERE NOT EXISTS (
    SELECT p.pid FROM parts p
    WHERE p.color = 'Green'
    EXCEPT
    SELECT c.pid FROM catalog c
    WHERE c.sid = s.sid
)
```
Join Expressions

- SQL keywords for operations we already saw

Cross Product:

Sailors **CROSS JOIN** Reserves

Condition Join:

Sailors **JOIN** Reserves **ON** <condition>

Natural Join: uses same-named columns for join columns

Sailors **NATURAL JOIN** Reserves

Usage Example:

SELECT *
FROM Sailors **JOIN** Reserves **ON** Sailors.sid=Reserves.sid
The syntax for an inner join that uses table aliases

```
SELECT select_list
FROM table_1 n1
    [INNER] JOIN table_2 n2
        ON n1.column_name operator n2.column_name
    [INNER] JOIN table_3 n3
        ON n2.column_name operator n3.column_name
...    
```

Note: The INNER keyword is optional and doesn’t change the meaning or action of the join. It just emphasizes that the JOIN is not an OUTER JOIN, coming up soon…
An inner join example from Murach Chap. 4

```
SELECT invoice_number, vendor_name, invoice_due_date,
    (invoice_total - payment_total - credit_total) AS balance_due
FROM vendors v JOIN invoices i
    ON v.vendor_id = i.vendor_id
WHERE (invoice_total - payment_total - credit_total) > 0
ORDER BY invoice_due_date DESC
```

The result set

<table>
<thead>
<tr>
<th>INVOICE_NUMBER</th>
<th>VENDOR_NAME</th>
<th>INVOICE_DUE_DATE</th>
<th>BALANCE_DUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 40318</td>
<td>Data Reproductions Corp</td>
<td>20-JUL-14</td>
<td>21842</td>
</tr>
<tr>
<td>2 39104</td>
<td>Data Reproductions Corp</td>
<td>20-JUL-14</td>
<td>85.31</td>
</tr>
<tr>
<td>3 0-2436</td>
<td>Malloy Lithographing Inc</td>
<td>17-JUL-14</td>
<td>10976.06</td>
</tr>
</tbody>
</table>

(40 rows selected)

Note: uses ORDER BY, not covered in R&G Chap. 5, but should be.

SQL in [fig4-02a.sql](#) (first load the needed tables with [create_ap_tables.sql](#))
A join with a table from another schema

```sql
SELECT vendor_name, customer_last_name, 
customer_first_name, vendor_state AS state, 
vendor_city AS city 
FROM vendors v 
JOIN om.customers c 
ON v.vendor_zip_code = c.customer_zip 
ORDER BY state, city
```

The result set

<table>
<thead>
<tr>
<th>VENDOR_NAME</th>
<th>CUSTOMER_LAST_NAME</th>
<th>CUSTOMER_FIRST_NAME</th>
<th>STATE</th>
<th>CITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wells Fargo Bank</td>
<td>Marissa</td>
<td>Kyle</td>
<td>AZ</td>
<td>Phoenix</td>
</tr>
<tr>
<td>Aztek Label</td>
<td>Irvin</td>
<td>Ana</td>
<td>CA</td>
<td>Anaheim</td>
</tr>
<tr>
<td>Lou Gentile’s Flower Basket</td>
<td>Damien</td>
<td>Deborah</td>
<td>CA</td>
<td>Fresno</td>
</tr>
<tr>
<td>Shields Design</td>
<td>Damien</td>
<td>Deborah</td>
<td>CA</td>
<td>Fresno</td>
</tr>
<tr>
<td>Costco</td>
<td>Neftaly</td>
<td>Thalia</td>
<td>CA</td>
<td>Fresno</td>
</tr>
<tr>
<td>Costco</td>
<td>Holbrooke</td>
<td>Rashad</td>
<td>CA</td>
<td>Fresno</td>
</tr>
<tr>
<td>Gary McKeighan Insurance</td>
<td>Holbrooke</td>
<td>Rashad</td>
<td>CA</td>
<td>Fresno</td>
</tr>
<tr>
<td>Zylka Design</td>
<td>Neftaly</td>
<td>Thalia</td>
<td>CA</td>
<td>Fresno</td>
</tr>
<tr>
<td>Zylka Design</td>
<td>Holbrooke</td>
<td>Rashad</td>
<td>CA</td>
<td>Fresno</td>
</tr>
</tbody>
</table>

(37 rows)

Oracle: another schema is another user’s schema
Mysql: another schema is another database
An inner join with two conditions

```sql
SELECT invoice_number, invoice_date,
       invoice_total, line_item_amt
FROM invoices i JOIN invoice_line_items li
    ON (i.invoice_id = li.invoice_id) AND
       (i.invoice_total > li.line_item_amt)
ORDER BY invoice_number
```

The result set

```
(6 rows selected)
```

SQL in `fig4-04a.sql`
The same join with one condition in a WHERE clause

```
SELECT invoice_number, invoice_date, invoice_total, line_item_amt
FROM invoices i JOIN invoice_line_items li
  ON i.invoice_id = li.invoice_id
WHERE i.invoice_total > li.line_item_amt
ORDER BY invoice_number
```

The result set

<table>
<thead>
<tr>
<th>INVOICE_NUMBER</th>
<th>INVOICE_DATE</th>
<th>INVOICE_TOTAL</th>
<th>LINE_ITEM_AMT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 97/522</td>
<td>30-APR-14</td>
<td>1962.13</td>
<td>765.13</td>
</tr>
<tr>
<td>2 97/522</td>
<td>30-APR-14</td>
<td>1962.13</td>
<td>1197</td>
</tr>
<tr>
<td>3 I77271-001</td>
<td>05-JUN-14</td>
<td>662</td>
<td>75.6</td>
</tr>
<tr>
<td>4 I77271-001</td>
<td>05-JUN-14</td>
<td>662</td>
<td>58.4</td>
</tr>
</tbody>
</table>

(6 rows selected)

SQL in `fig4-04b.sql`
A self-join that returns vendors from cities in common with other vendors

```sql
SELECT DISTINCT v1.vendor_name, v1.vendor_city, v1.vendor_state
FROM vendors v1 JOIN vendors v2
ON (v1.vendor_city = v2.vendor_city) AND
   (v1.vendor_state = v2.vendor_state) AND
   (v1.vendor_id <> v2.vendor_id)
ORDER BY v1.vendor_state, v1.vendor_city
```

The result set

84 rows selected

SQL in fig4-05.sql
A SELECT statement that joins four tables

```
SELECT vendor_name, invoice_number, invoice_date,
       line_item_amt, account_description
FROM vendors v
  JOIN invoices i ON v.vendor_id = i.vendor_id
  JOIN invoice_line_items li
    ON i.invoice_id = li.invoice_id
  JOIN general_ledger_accounts gl
    ON li.account_number = gl.account_number
WHERE (invoice_total - payment_total - credit_total) > 0
ORDER BY vendor_name, line_item_amt DESC
```

The result set

```
<table>
<thead>
<tr>
<th>VENDOR_NAME</th>
<th>INVOICE_NUMBER</th>
<th>INVOICE_DATE</th>
<th>LINE_ITEM_AMT</th>
<th>ACCOUNT_DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abbey Office Furnishings</td>
<td>203339-13</td>
<td>02-MAY-14</td>
<td>17.5</td>
<td>Office Supplies</td>
</tr>
<tr>
<td>Blue Cross</td>
<td>547481328</td>
<td>20-MAY-14</td>
<td>224</td>
<td>Group Insurance</td>
</tr>
<tr>
<td>Blue Cross</td>
<td>547480102</td>
<td>19-MAY-14</td>
<td>224</td>
<td>Group Insurance</td>
</tr>
<tr>
<td>Blue Cross</td>
<td>547479217</td>
<td>17-MAY-14</td>
<td>116</td>
<td>Group Insurance</td>
</tr>
<tr>
<td>Cardinal Business Media, Inc.</td>
<td>134116</td>
<td>01-JUN-14</td>
<td>90.36</td>
<td>Card Deck Advertising</td>
</tr>
<tr>
<td>Coffee Break Service</td>
<td>109596</td>
<td>14-JUN-14</td>
<td>41.8</td>
<td>Meals</td>
</tr>
<tr>
<td>Computerve</td>
<td>21-4748363</td>
<td>09-MAY-14</td>
<td>9.95</td>
<td>Books, Dues, and Subscriptions</td>
</tr>
<tr>
<td>Computerworld</td>
<td>367447</td>
<td>31-MAY-14</td>
<td>2433</td>
<td>Card Deck Advertising</td>
</tr>
</tbody>
</table>
```

(44 rows selected)

SQL in fig4-06.sql
A SELECT statement that joins four tables

```
SELECT vendor_name, invoice_number, invoice_date,
       line_item_amt, account_description
FROM vendors v
  JOIN invoices i ON v.vendor_id = i.vendor_id
  JOIN invoice_line_items li
    ON i.invoice_id = li.invoice_id
  JOIN general_ledger_accounts gl
    ON li.account_number = gl.account_number
WHERE (invoice_total - payment_total - credit_total) > 0
ORDER BY vendor_name, line_item_amt DESC
```

See how the joins follow the FKs here:
• FK invoices to vendors: vendor for invoice
• FK line_items to invoice: invoice for line_item
• FK line_item to general_ledger_accounts: account for line_item
Outer Joins: next time

- Include in join result non-matching tuples

- Result tuple padded with NULL values

- Variants
  - FULL: non-matching tuples in both relations included in result
  - LEFT: only non-matching tuples in left relation included in result
  - RIGHT: only non-matching tuples in right relation included in result