**RA/SQL Division, SQL JOINs**

Division: \( \div \) 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 \( y \):

  \[
  A \div B = \{ x \mid \exists y \in B \forall \langle x, y \rangle \in A \}
  \]

- \( A \div B \) contains all \( x \) tuples (sailors) such that for every \( y \) tuple (boat) in \( B \), there is an \( x \) 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 \( A \div B \).

- In general, \( x \) and \( y \) can be any sets of fields (not just singletons, but we won’t cover those cases)

**Visualizing Relationships**

- Sailors: sid1, sid2, sid3, sid4
- Reserves: bid1, bid2, bid3
- Boats: bid1, bid2, bid3, bid4

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 \( A \div B \) = table containing only sid sid2, the answer to the query about which sailors reserved all boats.

**Examples of Division \( A \div B \)**

<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>s2</td>
<td>b3</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>
</tbody>
</table>

\( A \div B \) 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>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>b1</td>
<td>b2</td>
<td>b1</td>
</tr>
<tr>
<td>s1</td>
<td>b2</td>
<td>B1</td>
<td>b4</td>
</tr>
<tr>
<td>s1</td>
<td>b3</td>
<td>B2</td>
<td>b4</td>
</tr>
<tr>
<td>s1</td>
<td>b4</td>
<td>B3</td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>s1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s2</td>
<td>s2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s3</td>
<td>s3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>s4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>s4</td>
<td>s1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>A/B1</td>
<td>A/B2</td>
<td>A/B3</td>
</tr>
</tbody>
</table>

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
  \[ A = \pi_{sid,bid} \text{Reserves} \]
  and table of bids is
  \[ B = \pi_{bid} \text{Boats} \]
  So the division is
  \[ (\pi_{sid,bid} \text{Reserves}) / (\pi_{bid} \text{Boats}) \]
- Providing the set of sids of sailors who reserved all the boats.

Query 1

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

\[ \rho(\text{Temp}\text{sid}, (\pi_{sid,bid} \text{Reserves}) / (\pi_{bid} \text{Boats})) \]
\[ \pi_{s\text{name}} (\text{Temp}\text{sid} \bowtie \text{Sailors}) \]

Query 2

“Find sailors who’ve reserved all red boats”

\[ \rho(\text{Temp}, (\pi_{sid,bid} \text{Reserves}) / (\pi_{bid} (\sigma_{\text{color} = \text{\textquoteleft red\textquoteright}} \text{Boats}))) \]
\[ \pi_{s\text{name}} (\text{Temp} \bowtie \text{Sailors}) \]

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:
  \[ \pi_{x} (A) \bowtie B \]
  \[ \pi_{x} (A) \bowtie B = A \]

Disqualified x values:

\[ \pi_{x} ((\pi_{x} (A) \bowtie B) \bowtie A) \]

A/B:

\[ \pi_{x} (A) \bowtie \text{all disqualified tuples} \]
\[ \pi_{x} (A) \bowtie \pi_{x} ((\pi_{x} (A) \bowtie B) \bowtie A) \]

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>s2</td>
<td>b1</td>
</tr>
<tr>
<td>s2</td>
<td>b2</td>
</tr>
<tr>
<td>s3</td>
<td>b2</td>
</tr>
<tr>
<td>s3</td>
<td>b3</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, b2>, so x=s2 is disqualified from A/B.

\[ \pi_{x} (A) = (s1, s2, s3, s4) \]
\[ \pi_{x} (A) \bowtie B = ((s1, b1), (s1, b2), (s1, b3), (s1, b4)) \] (16 tuples)
\[ (\pi_{x} (A) \bowtie B) \bowtie A = ((s2, b2), (s2, b3), (s2, b4), ...) \]
\[ (\pi_{x} (A) \bowtie B) \bowtie A = (s2, s3, s4) \]
\[ = \text{all disqualified tuples, leaving only s1 qualified} \]
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)

Division: Solution 1 (Q9, pg. 150)

“Find sailors who’ve reserved all boats.”

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

```sql
SELECT S.name
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 (…)

“Find sailors who’ve reserved all boats.”

Without `EXCEPT`: so works on mysql

```sql
SELECT S.name
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 )
  )
```

Division: Solution 2 (also on pg. 150)

“Find sailors who’ve reserved all boats.”

Without `EXCEPT`: so works on mysql

```sql
SELECT S.name
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 )
  )
```

Division: Solution 3 (not in text)

“Find sailors who’ve reserved all boats.”

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

The HAVING group for a certain sid qualifies the sid if it contains all the bids, shown by proper count.

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 EXCEPT:

```
SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (
    (SELECT B.bid FROM Boats B
     WHERE B.color = 'red')
    EXCEPT
    (SELECT R.bid FROM Reserves R
     WHERE R.sid = S.sid)
)
```

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

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

```
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)
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)
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
)
```

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**

```sql
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**

```sql
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 0016</td>
<td>Data Reproduction Co.</td>
<td>20-07-14</td>
<td>2794</td>
</tr>
<tr>
<td>2 0018</td>
<td>Data Reproduction Co.</td>
<td>20-07-14</td>
<td>0.33</td>
</tr>
<tr>
<td>3 1249</td>
<td>Sally's Importing Inc.</td>
<td>17-07-14</td>
<td>1070.00</td>
</tr>
</tbody>
</table>

(40 rows selected)

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

SQL in [fig4-03.a.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>Multi File Host</td>
<td>Holmes</td>
<td>Rich</td>
<td>CA</td>
<td>Bakers</td>
</tr>
<tr>
<td>J &amp; J Tools</td>
<td>Song</td>
<td>Jean</td>
<td>GB</td>
<td>Cessna</td>
</tr>
<tr>
<td>Car Dealers</td>
<td>beetle</td>
<td>Mary</td>
<td>GB</td>
<td>Fredo</td>
</tr>
<tr>
<td>Printers</td>
<td>Leslie</td>
<td>John</td>
<td>GB</td>
<td>Frisco</td>
</tr>
<tr>
<td>Tools</td>
<td>Lesley</td>
<td>John</td>
<td>GB</td>
<td>Frezno</td>
</tr>
<tr>
<td>Home Print</td>
<td>Michelle</td>
<td>Terry</td>
<td>GB</td>
<td>Frezno</td>
</tr>
<tr>
<td>Office Depot</td>
<td>Michelle</td>
<td>Terry</td>
<td>GB</td>
<td>Frezno</td>
</tr>
<tr>
<td>Office Supply</td>
<td>Michelle</td>
<td>Terry</td>
<td>GB</td>
<td>Frezno</td>
</tr>
</tbody>
</table>

(37 rows)

Oracle: another schema is another user's schema

Mysql: another schema is another database

SQL in [fig4-04.a.sql]

**The same join with one condition in a WHERE clause**

```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
WHERE i.invoice_total > li.invoice_id
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 0016</td>
<td>20-07-14</td>
<td>2794</td>
<td>0.33</td>
</tr>
<tr>
<td>2 0018</td>
<td>20-07-14</td>
<td>0.33</td>
<td>2794</td>
</tr>
<tr>
<td>3 1249</td>
<td>17-07-14</td>
<td>1070.00</td>
<td>0.33</td>
</tr>
</tbody>
</table>

(6 rows selected)

SQL in [fig4-04.b.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)
ORDER BY v1.vendor_city
```

The result set

<table>
<thead>
<tr>
<th>vendor_name</th>
<th>vendor_city</th>
<th>vendor_state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi File Host</td>
<td>Bakers</td>
<td>CA</td>
</tr>
<tr>
<td>J &amp; J Tools</td>
<td>Cessna</td>
<td>GB</td>
</tr>
<tr>
<td>Car Dealers</td>
<td>Frisco</td>
<td>GB</td>
</tr>
<tr>
<td>Printers</td>
<td>Frisco</td>
<td>GB</td>
</tr>
<tr>
<td>Tools</td>
<td>Frisco</td>
<td>GB</td>
</tr>
<tr>
<td>Home Print</td>
<td>Frezno</td>
<td>GB</td>
</tr>
<tr>
<td>Office Depot</td>
<td>Frezno</td>
<td>GB</td>
</tr>
<tr>
<td>Office Supply</td>
<td>Frezno</td>
<td>GB</td>
</tr>
<tr>
<td>Office Supply</td>
<td>Frezno</td>
<td>GB</td>
</tr>
</tbody>
</table>

(84 rows selected)

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

```sql
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 Amount</th>
<th>Account Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC Corp</td>
<td>12345</td>
<td>01-01-2023</td>
<td>12345</td>
<td>Accounting Services</td>
</tr>
<tr>
<td>DEF Corp</td>
<td>67890</td>
<td>02-02-2023</td>
<td>67890</td>
<td>Legal Services</td>
</tr>
<tr>
<td>GHI Corp</td>
<td>34567</td>
<td>03-03-2023</td>
<td>34567</td>
<td>Consulting Services</td>
</tr>
<tr>
<td>JKL Corp</td>
<td>78901</td>
<td>04-04-2023</td>
<td>78901</td>
<td>Creative Services</td>
</tr>
</tbody>
</table>

(44 rows selected)

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