Queries

- Information is extracted from a database by writing *queries*
  - query inputs: relations
  - query result: relation

- queries are *read-only* operations

- The SQL command for queries *select*
Example SQL Query

```
SELECT name
FROM Customer
WHERE zipcode='95211'
```

attributes to extract

input relations

conditions used to filter tuples
The Relational Algebra

- The *relational algebra* defines mathematical operations on relations
  - provides a solid theory for dealing with queries

- These operations define the meaning of SQL queries
  - *think* in terms of the algebra operators
  - *translate* the algebra to SQL syntax

- Query optimization:
  - SQL queries are translated to expression trees containing algebraic operators
  - expression trees are reorganized to optimize the order of operations
Relational Algebra Operations

- set theoretic:
  - **union, intersection, difference, cross-product**
  - usual mathematical meaning

- relational database unique:
  - **select, project**: extraction from a single relation
  - **join**: combining two relations
The SELECT Operation

- SELECT extracts tuples from a relation
  - result has same relation schema as operand
- SELECT requires a selection condition
  - selection condition is a boolean expression to filter tuple values
- Syntax:

\[ \sigma_{<\text{selection condition}>} (R) \]

- Selection condition may contain
  AND, OR, NOT, =, <, ≤, >, ≥, ≠
SELECT Examples

\[ r_2(\text{STORESTOCK}) = \begin{cases} 
< "S002", "I065", 120 >, \\
< "S333", "I954", 198 >, \\
< "S047", "I099", 267 >, \\
< "S047", "I954", 300 > 
\end{cases} \]

\[ \sigma_{\text{StoreId} = "S047"}(\text{STORESTOCK}) = \begin{cases} 
< "S047", "I099", 267 >, \\
< "S047", "I954", 300 > 
\end{cases} \]

\[ \sigma_{\text{quantity} < 200}(\text{STORESTOCK}) = \begin{cases} 
< "S002", "I065", 120 >, \\
< "S333", "I954", 198 > 
\end{cases} \]
The PROJECT Operation

- PROJECT extracts attributes from a relation
  - result schema attributes are a subset of the operand schema

- PROJECT requires a attribute list

- Syntax:

\[ \pi_{\text{attribute list}}(R) \]

- Duplicates are not kept in result
  - result is a relation, which is a set

\[ \pi_{\text{StoreId, Item}}(\text{STORESTOCK}) = \{ < "S002", "l065" >, < "S333", "l954" >, < "S047", "l099" >, < "S047", "l954" > \} \]
PROJECT Examples

\[ r(\text{STOCKITEM}) = \{ <"I075", "Ice Cream", $1.49, false >, <"I345", "Cupcakes", $1.99, false >, <"I333", "Twinkies", $1.98, false > \} \]

\[ \pi_{\text{Description, Price}}(\text{STOREITEM}) = \{ <"Ice Cream", $1.49 >, <"Cupcakes", $1.99 >, <"Twinkies", $1.98 > \} \]
Composing Operations

\[ r(\text{STOCKITEM}) = \begin{cases} < "I075", "Ice Cream", $1.49, \text{false} >, \\ < "I345", "Cupcakes", $1.99, \text{false} >, \\ < "I333", "Twinkies", $1.98, \text{false} > \end{cases} \]

\[ \pi_{\text{Price}} (\sigma_{\text{Description}="Ice Cream"}(\text{STOREITEM})) \]

SELECT result: \( \begin{cases} < "I075", "Ice Cream", $1.49, \text{false} > \end{cases} \)

PROJECT result: \( \begin{cases} < $1.49 > \end{cases} \)
The JOIN Operation

- JOIN combines tuples from two tables based on values of related attributes (usually a FK)

- JOIN requires a join condition
  - boolean expression comparing attributes from each operand

- Syntax:

  \[ R \bowtie_{<\text{join condition}>} S \]

- The join condition may contain
  \[ \text{AND, } =, \leq, \geq, \neq \]
JOIN Examples

STORESTOCK( StoreId, Item, Quantity )
STOCKITEM( ItemId, Description, Price, Taxable )

r(STOCKITEM) = \{ <"I075", "Ice Cream", $1.49, false >, 
                 <"I345", "Cupcakes", $1.99, false >, 
                 <"I333", "Twinkies", $1.98, false > \}

r(STORESTOCK) = \{ <"S002", "I075", 120 >, 
                    <"S047", "I333", 267 > \}

STORESTOCK $\bowtie$ \langle \text{Item = ItemId} \rangle$ STOCKITEM

\[ \text{STORESTOCK( StoreId, Item, Quantity )} \]
\[ \text{STOCKITEM( ItemId, Description, Price, Taxable )} \]
\[ r(\text{STOCKITEM}) = \{ <"I075", "Ice Cream", $1.49, false >, <"I345", "Cupcakes", $1.99, false >, <"I333", "Twinkies", $1.98, false > \} \]
\[ r(\text{STORESTOCK}) = \{ <"S002", "I075", 120 >, <"S047", "I333", 267 > \} \]

\[ \text{STORESTOCK} \bowtie \langle \text{Item = ItemId} \rangle \text{ STOCKITEM} \]
JOIN Examples

\[
\text{RESULT}(\text{StoreId}, \text{Item}, \text{Quantity}, \text{ItemId}, \text{Description}, \text{Price}, \text{Taxable})
\]

\[
r(\text{RESULT}) = \left\{ <"I075", "Ice Cream", $1.49, false, "S002", "I075", 120 >, \\
<"I333", "Twinkies", $1.98, false, "S047", "I333", 267 > \right\}
\]

\[
\text{STORESTOCK} \bowtie_{\text{Item} = \text{ItemId}} \text{STOCKITEM}
\]
**Query:**
Get the Description and Price for all Items stocked by Store S002

<table>
<thead>
<tr>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
</tr>
</thead>
<tbody>
<tr>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
</tr>
<tr>
<td>I345</td>
<td>Cup Cakes</td>
<td>$1.99</td>
<td>FALSE</td>
</tr>
<tr>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Result has these attributes

Result only includes tuples with certain ItemIds
**Query:** Get the Description and Price for all Items stocked by Store S002

### STORESTOCK

<table>
<thead>
<tr>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>S002</td>
<td>I075</td>
<td>120</td>
</tr>
<tr>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
<tr>
<td>S002</td>
<td>I333</td>
<td>1200</td>
</tr>
</tbody>
</table>

### STOCKITEM

<table>
<thead>
<tr>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
</tr>
</thead>
<tbody>
<tr>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
</tr>
<tr>
<td>I345</td>
<td>Cup Cakes</td>
<td>$1.99</td>
<td>FALSE</td>
</tr>
<tr>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

We need a join to merge data across relations.
Writing Queries

```
STORESTOCK ⋈<Item = ItemId> STOCKITEM
```

<table>
<thead>
<tr>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
</tr>
</thead>
<tbody>
<tr>
<td>S002</td>
<td>I075</td>
<td>120</td>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
</tr>
<tr>
<td>S047</td>
<td>I333</td>
<td>267</td>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
</tr>
<tr>
<td>S002</td>
<td>I333</td>
<td>1200</td>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Query:
Get the Description and Price for all Items stocked by Store S002

Now we can select and project to extract the information we want
Writing Queries

\[ R_1 = \text{STORESTOCK} \bowtie_{\text{Item} = \text{ItemId}} \text{STOCKITEM} \]

<table>
<thead>
<tr>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
</tr>
</thead>
<tbody>
<tr>
<td>S002</td>
<td>1075</td>
<td>120</td>
<td>1075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
</tr>
<tr>
<td>S047</td>
<td>1333</td>
<td>267</td>
<td>1333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
</tr>
<tr>
<td>S002</td>
<td>1333</td>
<td>1200</td>
<td>1333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

\[ R_2 = \sigma_{\text{StoreId} = "S002"}(R_1) \]

<table>
<thead>
<tr>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
</tr>
</thead>
<tbody>
<tr>
<td>S002</td>
<td>1075</td>
<td>120</td>
<td>1075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
</tr>
<tr>
<td>S002</td>
<td>1333</td>
<td>1200</td>
<td>1333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
</tr>
</tbody>
</table>
Writing Queries

\[ R_2 = \sigma_{\text{StoreId} = "S002"} (R_1) \]

<table>
<thead>
<tr>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
</tr>
</thead>
<tbody>
<tr>
<td>S002</td>
<td>I075</td>
<td>120</td>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
</tr>
<tr>
<td>S002</td>
<td>I333</td>
<td>1200</td>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

\[ R_3 = \pi_{\langle \text{Description, Price} \rangle} (R_2) \]

<table>
<thead>
<tr>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ice Cream</td>
<td>$1.49</td>
</tr>
<tr>
<td>Twinkies</td>
<td>$1.98</td>
</tr>
</tbody>
</table>
Since the operands and results of every query are relations, we can compose or chain queries.

\[ R_1 = \text{STORESTOCK} \bowtie_{\langle \text{Item} = \text{ItemId} \rangle} \text{STOCKITEM} \]
\[ R_2 = \sigma_{\text{StoreId} = "S002"}(R_1) \]
\[ R_3 = \pi_{\langle \text{Description, Price} \rangle}(R_2) \]

\[ \pi_{\langle \text{Description, Price} \rangle}(\sigma_{\text{StoreId} = "S002"}(\text{STORESTOCK} \bowtie_{\langle \text{Item} = \text{ItemId} \rangle} \text{STOCKITEM})) \]
Relational algebra queries are easily translated to SQL queries (more on this later)

\[ \pi_{\text{Description, Price}}\left(\sigma_{\text{StoreId} = "S002"} \left(\text{STORESTOCK} \bowtie_{\text{Item} = \text{ItemId}} \text{STOCKITEM}\right)\right) \]

```
SELECT Description, Price
FROM   STORESTOCK, STOCKITEM
WHERE  StoreId='S002'
      AND STORESTOCK.Item = STOCKITEM.ItemId
```
EXERCISE 1: Schema

**EMPLOYEE**
- Fname
- Minit
- Lname
- Ssn
- Bdate
- Address
- Sex
- Salary
- Super_ssn
- Dno

**DEPARTMENT**
- Dname
- Dnumber
- Mgr_ssn
- Mgr_start_date

**DEPT_LOCATIONS**
- Dnumber
- Dlocation

**PROJECT**
- Pname
- Pnumber
- Plocation
- Dnum

**WORKS_ON**
- Essn
- Pno
- Hours

**DEPENDENT**
- Essn
- Dependent_name
- Sex
- Bdate
- Relationship
EXERCISE 1: Queries

Express the following queries in the algebra:

1. First and last name of employees who have no supervisor.
2. First and last name of employees supervised by Franklin Wong.
3. Last name of employees who have dependents.
4. Last name of employees who have daughters.
5. Last name of employees in department 5 who work more than 10 hours/week on ProductX.
6. Last name of supervisors of employees in department 5 who work more than 10 hours/week on ProductX.
Set Operations

- **UNION**: $R \cup S$
  - all tuples in either $R$ or $S$

- **INTERSECTION**: $R \cap S$
  - all tuples in both $R$ and $S$

- **DIFFERENCE**: $R - S$
  - all tuples in $R$ but not in $S$

- **CROSS-PRODUCT or CARTESIAN PRODUCT**: $R \times S$
  - pair all tuples in $R$ with all tuples in $S$

**$\cup$, $\cap$**
- operands must be union compatible
  - (same attribute types)
# UNION: example

**STORESTOCK**

<table>
<thead>
<tr>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>S002</td>
<td>1075</td>
<td>120</td>
</tr>
<tr>
<td>S047</td>
<td>1333</td>
<td>267</td>
</tr>
<tr>
<td>S002</td>
<td>1333</td>
<td>267</td>
</tr>
</tbody>
</table>

**WAREHOUSESTOCK**

<table>
<thead>
<tr>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>W998</td>
<td>1075</td>
<td>1200</td>
</tr>
<tr>
<td>W087</td>
<td>1001</td>
<td>5000</td>
</tr>
<tr>
<td>W222</td>
<td>1188</td>
<td>11500</td>
</tr>
<tr>
<td>W023</td>
<td>1075</td>
<td>300</td>
</tr>
</tbody>
</table>

**STORESTOCK U WAREHOUSESTOCK**

<table>
<thead>
<tr>
<th>Id</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>S002</td>
<td>1075</td>
<td>120</td>
</tr>
<tr>
<td>S047</td>
<td>1333</td>
<td>267</td>
</tr>
<tr>
<td>S002</td>
<td>1333</td>
<td>267</td>
</tr>
<tr>
<td>W998</td>
<td>1075</td>
<td>1200</td>
</tr>
<tr>
<td>W087</td>
<td>1001</td>
<td>5000</td>
</tr>
<tr>
<td>W222</td>
<td>1188</td>
<td>11500</td>
</tr>
<tr>
<td>W023</td>
<td>1075</td>
<td>300</td>
</tr>
</tbody>
</table>
INTERSECTION: example

\[ \pi_{ItemID}(STORESTOCK) \cap \pi_{Item}(WAREHOUSESTOCK) = \]

\[
\begin{array}{c|c|c}
\text{Item} & \text{I075} & \text{I333} \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{Item} & \text{I075} & \text{I001} & \text{I188} \\
\hline
\end{array}
\]
**DIFERENCE: example**

<table>
<thead>
<tr>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
</tr>
</thead>
<tbody>
<tr>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
</tr>
<tr>
<td>I345</td>
<td>Cup Cakes</td>
<td>$1.99</td>
<td>FALSE</td>
</tr>
<tr>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>S002</td>
<td>I075</td>
<td>120</td>
</tr>
<tr>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
<tr>
<td>S002</td>
<td>I333</td>
<td>267</td>
</tr>
</tbody>
</table>

\[
\pi_{\text{itemId}}(\text{STOCKITEM}) - \pi_{\text{item}}(\text{STORESTOCK})
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{itemId} & \text{Item} \\
\hline
I075 & \hline
I345 & \hline
I333 & \hline
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\text{Item} \\
\hline
I075 & \hline
I333 & \hline
I345 & \hline
\hline
\end{array}
\]

\[
= \begin{array}{|c|}
\hline
\text{Item} \\
\hline
I345 & \hline
\hline
\end{array}
\]
**CROSS PRODUCT: example**

<table>
<thead>
<tr>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
</tr>
</thead>
<tbody>
<tr>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
</tr>
<tr>
<td>I345</td>
<td>Cup Cakes</td>
<td>$1.99</td>
<td>FALSE</td>
</tr>
<tr>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**STORESTOCK**

<table>
<thead>
<tr>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>S002</td>
<td>I075</td>
<td>120</td>
</tr>
<tr>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
</tbody>
</table>

**STOCKITEM x STORESTOCK**

<table>
<thead>
<tr>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
<td>S002</td>
<td>I075</td>
<td>120</td>
</tr>
<tr>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
<tr>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
<tr>
<td>I345</td>
<td>Cup Cakes</td>
<td>$1.99</td>
<td>FALSE</td>
<td>S002</td>
<td>I075</td>
<td>120</td>
</tr>
<tr>
<td>I345</td>
<td>Cup Cakes</td>
<td>$1.99</td>
<td>FALSE</td>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
<tr>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
</tbody>
</table>
CROSS PRODUCT

- CROSS PRODUCT is also called CROSS JOIN
  - a JOIN with no join condition

- XPROD brings together all possible information from two relations

- XPROD can result in very large relations
  - if R has i tuples and S has j tuples, then R \( \times \) S has i\( \times \)j tuples

- In real applications, avoid XPROD whenever possible
  - Database equivalent of a brute-force nested loop
  - massive unnecessary memory usage
JOIN = XPROD and SELECT

STORESTOCK ⋈<Item = ItemId> STOCKITEM

<table>
<thead>
<tr>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
<td>S002</td>
<td>I075</td>
<td>120</td>
</tr>
<tr>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
</tbody>
</table>

σ_{ItemId=Item} (STOCKITEM × STORESTOCK)

<table>
<thead>
<tr>
<th>ItemId</th>
<th>Description</th>
<th>Price</th>
<th>Taxable</th>
<th>StoreId</th>
<th>Item</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
<td>S002</td>
<td>I075</td>
<td>120</td>
</tr>
<tr>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
<tr>
<td>I345</td>
<td>Cup Cakes</td>
<td>$1.99</td>
<td>FALSE</td>
<td>S002</td>
<td>I075</td>
<td>120</td>
</tr>
<tr>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
<tr>
<td>I075</td>
<td>Ice Cream</td>
<td>$1.49</td>
<td>FALSE</td>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
<tr>
<td>I345</td>
<td>Cup Cakes</td>
<td>$1.99</td>
<td>FALSE</td>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
<tr>
<td>I333</td>
<td>Twinkies</td>
<td>$1.98</td>
<td>FALSE</td>
<td>S047</td>
<td>I333</td>
<td>267</td>
</tr>
</tbody>
</table>
Relational algebra queries are easily translated to SQL queries

\[ \pi_{\langle \text{Description, Price} \rangle} \left( \sigma_{\text{StoreId} = "S002"} \left( \text{STORESTOCK} \bowtie_{\langle \text{Item} = \text{ItemId} \rangle} \text{STOCKITEM} \right) \right) \]

```
SELECT Description, Price
FROM   STORESTOCK, STOCKITEM
WHERE  StoreId='S002'
AND    STORESTOCK.Item = STOCKITEM.ItemId
```
Queries are **declarative**, not **procedural**.

These queries specify the information we want, in terms of mathematical operations … but not how to compute it.

Does not say “compute a cross-product, then select …”, rather give me something “equivalent to cross-product, select …”

\[
\pi_{\langle \text{Description, Price} \rangle} (\sigma_{\text{StoreId} = "S002"} (\text{STORESTOCK} \bowtie_{\langle \text{Item} = \text{ItemId} \rangle} \text{STOCKITEM}))
\]

SELECT Description, Price
FROM STORESTOCK, STOCKITEM
WHERE StoreId='S002'
AND STORESTOCK.Item = STOCKITEM.ItemId
A complete set of operations is a subset of all operations that is sufficient to express all queries expressible by all operators. Any operators outside a complete set can be expressed as combinations of operators in the complete set.

Complete Set: \{ \sigma, \pi, \cup, -, \times \}

Operators outside a complete set are not necessary, but they are generally convenient.

- JOIN: \( A \bowtie C B = \sigma_C(A \times B) \)
- INTERSECTION: \( A \cap B = (A \cup B) - ((A - B) \cup (B - A)) \)
There are some queries in commercial languages (i.e. SQL) that cannot be expressed in the basic relational algebra

A few additional operations handle most of these

- **aggregation**: apply a function to a collection of values

- **OUTER JOINS** and **OUTER UNIONS**: null values are used to keep data that would otherwise be discarded
<table>
<thead>
<tr>
<th>Operation</th>
<th>Purpose</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Selects all tuples that satisfy the selection condition from a relation $R$.</td>
<td>$\sigma_{&lt;\text{selection condition}&gt;}(R)$</td>
</tr>
<tr>
<td>PROJECT</td>
<td>Produces a new relation with only some of the attributes of $R$, and removes duplicate tuples.</td>
<td>$\pi_{&lt;\text{attribute list}&gt;}(R)$</td>
</tr>
<tr>
<td>THETA JOIN</td>
<td>Produces all combinations of tuples from $R_1$ and $R_2$ that satisfy the join condition.</td>
<td>$R_1 \bowtie_{&lt;\text{join condition}&gt;} R_2$</td>
</tr>
<tr>
<td>EQUIJOIN</td>
<td>Produces all the combinations of tuples from $R_1$ and $R_2$ that satisfy a join condition with only equality comparisons.</td>
<td>$R_1 \bowtie_{&lt;\text{join condition}&gt;} R_2$, $\text{OR } R_1 \bowtie_{(&lt;\text{join attributes 1}&gt;),\text{(&lt;join attributes 2&gt;})} R_2$</td>
</tr>
<tr>
<td>NATURAL JOIN</td>
<td>Same as EQUIJOIN except that the join attributes of $R_2$ are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.</td>
<td>$R_1 \bowtie_{&lt;\text{join condition}&gt;} R_2$, $\text{OR } R_1 \bowtie_{(&lt;\text{join attributes 1}&gt;),\text{(&lt;join attributes 2&gt;})} R_2$, $\text{OR } R_1 \bowtie_{&lt;\text{join condition}&gt;} R_2$, $\text{OR } R_1 \bowtie_{(&lt;\text{join attributes 1}&gt;),\text{(&lt;join attributes 2&gt;})} R_2$</td>
</tr>
<tr>
<td>UNION</td>
<td>Produces a relation that includes all the tuples in $R_1$ or $R_2$ or both $R_1$ and $R_2$; $R_1$ and $R_2$ must be union compatible.</td>
<td>$R_1 \cup R_2$</td>
</tr>
<tr>
<td>INTERSECTION</td>
<td>Produces a relation that includes all the tuples in both $R_1$ and $R_2$; $R_1$ and $R_2$ must be union compatible.</td>
<td>$R_1 \cap R_2$</td>
</tr>
<tr>
<td>DIFFERENCE</td>
<td>Produces a relation that includes all the tuples in $R_1$ that are not in $R_2$; $R_1$ and $R_2$ must be union compatible.</td>
<td>$R_1 - R_2$</td>
</tr>
<tr>
<td>CARTESIAN PRODUCT</td>
<td>Produces a relation that has the attributes of $R_1$ and $R_2$ and includes as tuples all possible combinations of tuples from $R_1$ and $R_2$.</td>
<td>$R_1 \times R_2$</td>
</tr>
<tr>
<td>DIVISION</td>
<td>Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in $R_1$ in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.</td>
<td>$R_1(Z) \div R_2(Y)$</td>
</tr>
</tbody>
</table>
Query trees are representations of queries that can be manipulated by query optimizers, according to mathematical properties of the operators. (Chapter 15)
EXERCISE 2: Schema

EMPLOYEE
- Fname
- Minit
- Lname
- Ssn
- Bdate
- Address
- Sex
- Salary
- Super_ssn
- Dno

DEPARTMENT
- Dname
- Dnumber
- Mgr_ssn
- Mgr_start_date

DEPT_LOCATIONS
- Dnumber
- Dlocation

PROJECT
- Pname
- Pnumber
- Plocation
- Dnum

WORKS_ON
- Essn
- Pno
- Hours

DEPENDENT
- Essn
- Dependent_name
- Sex
- Bdate
- Relationship
EXERCISE 2: Queries

1. First and last names of all department managers.

2. Salaries of all employees who have worked on the Reorganization project.

3. SSN of all employees who have worked on a project that is controlled by a department different than the department that they are assigned to.

4. Last name of all employees who are not married.
# Exercise 3: Schema

<table>
<thead>
<tr>
<th>AIRPORT</th>
<th>Flight_number</th>
<th>Name</th>
<th>City</th>
<th>State</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>FLIGHT</th>
<th>Flight_number</th>
<th>Airline</th>
<th>Weekdays</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>FLIGHT_LEG</th>
<th>Flight_number</th>
<th>Leg_number</th>
<th>Departure_airport_code</th>
<th>Scheduled_departure_time</th>
<th>Arrival_airport_code</th>
<th>Scheduled_arrival_time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LEG_INSTANCE</th>
<th>Flight_number</th>
<th>Leg_number</th>
<th>Date</th>
<th>Number_of_available_seats</th>
<th>Airplane_id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FARE</th>
<th>Flight_number</th>
<th>Fare_code</th>
<th>Amount</th>
<th>Restrictions</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>AIRPLANE_TYPE</th>
<th>Airplane_type_name</th>
<th>Max_seats</th>
<th>Company</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CAN_LAND</th>
<th>Airplane_type_name</th>
<th>Airport_code</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>AIRPLANE</th>
<th>Airplane_id</th>
<th>Total_number_of_seats</th>
<th>Airplane_type</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SEAT_RESERVATION</th>
<th>Flight_number</th>
<th>Leg_number</th>
<th>Date</th>
<th>Seat_number</th>
<th>Customer_name</th>
<th>Customer_phone</th>
</tr>
</thead>
</table>
EXERCISE 3: Queries

1. List all airplane types that can land at any airport in San Francisco.
2. List the ids and number of seats for all airplanes that can land at any airport in Chicago.
3. List the name and phone number of all customers with a seat reserved on a flight that leaves Chicago O’Hara airport (ORD) on October 31, 2008.
4. List all airlines that have seats available for flights leaving Los Angeles (LAX) on September 25, 2008.
5. List all airlines that operate at San Jose International Airport (SJC).