Union

- Union-compatible
  - two relations are union-compatible if they have the same # of attributes and each attribute must be from the same domain
  - to compute $R_1 \cup R_2$, $R_1$ and $R_2$ must be union-compatible

- Order of relations unimportant

  $R_1 \cup R_2 = R_2 \cup R_1$
  $R_1 \cup R_2 = \{ r \in R_1 \lor r \in R_2 \}$

<ex>
$R_1$ (A, B) = \{a_1 b_1, a_2 b_1\}
$R_2$ (A, B) = \{a_1 b_1, a_4 b_2\}

$R_3 = R_1 \cup R_2$
$= \{a_1 b_1, a_2 b_1, a_4 b_2\}$
Set Difference

- $R_1$ and $R_2$ must be union-compatible
- $R_1 - R_2 = \{ r \mid r \in R_1 \land r \notin R_2 \}$
- Order of relation is important

<ex>
$R_1 (A, B) = \{ a_1 b_1, a_2 b_1 \}$
$R_2 (A, B) = \{ a_1 b_1, a_4 b_2 \}$
$R_3 = R_1 - R_2 = \{ a_2 b_1 \}$
</ex>
Intersection

- $R_1$ and $R_2$ must be union-compatible
- $R_1 \cap R_2 = \{ r | r \in R_1 \land r \in R_2 \}$
- Order of relation is unimportant

<ex>
$R_1 (A, B) = \{ a_1 b_1, a_2 b_1 \}$
$R_2 (A, B) = \{ a_1 b_1, a_4 b_2 \}$
$R_3 = R_1 \cap R_2 = \{ a_1 b_1 \}$
Cartesian Product

\[ R_1 \times R_2 = \{ <r_1, r_2> | r_1 \in R_1 \land r_2 \in R_2 \} \]

<ex>
\[ R_1 (A, B) = \{ a_1 b_1, a_2 b_1 \} \]
\[ R_2 (A, B) = \{ a_1 b_1, a_4 b_2 \} \]
\[ R_3 = R_1 \times R_2 \]
\[ = \{ a_1 b_1 a_1 b_1, \]
\[ a_1 b_1 a_4 b_2, \]
\[ a_2 b_1 a_1 b_1, \]
\[ a_2 b_1 a_4 b_2 \} \]
<ex>
Course = (C#, Cname, Dept)
Class = (C#, Semester, Prof)
Course \times Class = (C#, Cname, Dept, C#, Semester, Prof)
Selection

- $\sigma_F$:
  - $F$: formula (or predicate)
  - selection criteria can be either logical or arithmetic
  - $\sigma_F (R)$ creates a relation consisting of tuples in $R$
    for which $F(t)$ is true
  - $\sigma_F (R) = \{ t \mid t \in R \land F(t) \}$

- Selection is commutative

  \[ \sigma_{A=a} (\sigma_{B=b} (R)) = \sigma_{B=b} (\sigma_{A=a} (R)) = \sigma_{A=a, B=b} (R) \]

- Selection is distributive over binary operators

  \[ \sigma_F (R_1 \cup R_2) = \sigma_F (R_1) \cup \sigma_F (R_2) \]
  \[ \sigma_F (R_1 \cap R_2) = \sigma_F (R_1) \cap \sigma_F (R_2) \]
  \[ \sigma_F (R_1 - R_2) = \sigma_F (R_1) - \sigma_F (R_2) \]
Selection

Theorem:

\[ \sigma_F (R_1 \cap R_2) = \sigma_F (R_1) \cap \sigma_F (R_2) \]

<Proof>

\[ \sigma_F (R_1 \cap R_2) = \{ t \mid t \in R_1 \cap R_2 \land F(t) \} \]

\[ = \{ t \mid t \in R_1 \land t \in R_2 \land F(t) \} \]

\[ = \{ t \mid t \in R_1 \land F(t) \land t \in R_2 \land F(t) \} \]

\[ = \sigma_F (R_1) \cap \sigma_F (R_2) \]
Projection

- \( \Pi \)
- A vertical subset of a relation
- Eliminates some attributes from a relation and then eliminate duplicate tuples, if any.

<ex> Class (C#, Semester, Dept)  
-------------------------------------- 
PH 1 F93 PHIL 
PH 1 S94 PHIL 
CS662 S94 CS 
CS662 S93 CS 

\( \Pi_{C\#} \) (Class) = C\# \{PH1, CS662\}

- Projection commutes with selection, if attributes for selection are among attributes in the set unto which \( \Pi \) is taking place

\( \Pi_X(\sigma_F(R)) = \sigma_F(\Pi_X(R)) \)

if \( X \subseteq R \) and F is a valid expression in X
Join

- Central to relational database theory
  - used to combine related tuples from two relations into single tuples
- General form
  \[ R \times_{\text{join-condition}} S \]
  where \( R = (A_1, \ldots, A_n) \) and \( S = (B_1, \ldots, B_m) \)
  will result in \( Q = (A_1, \ldots, A_n, B_1, \ldots, B_m) \)
  - \( Q \) has one tuple for each combination of tuples (one from \( R \) and one from \( S \)), whenever the combination satisfy the join condition
- Difference from Cartesian products
  - in join, only combinations of tuples satisfying the join condition appear in the result
  - in Cartesian product, all combinations of tuples appear
Join

• Theta join
  - a join condition is of the form $A_i \Theta B_j$
  where $A_i$ is an attribute of $R$ and $B_j$ of $S$,
  and $A_i$ and $B_j$ have the same domain
  - $\Theta$ is one of the comparison operators:
    \{ $=$, $\neq$, $<$, $>$, $\geq$, $\leq$ \}
  - $R \mid \times \Theta S = \sigma_{\Theta} (R \times S)$
    $\Theta$: selection predicate

• Equijoin
  - if only comparison operator used is "=",
    it is equijoin
  - results of an equijoin always have one or more pairs of
    attributes that have identical values in every tuple
  - get rid of the second attribute: natural join
Natural Join

- An equijoin on all attributes with the same name in two relations, followed by removal of superfluous attributes
  
  1) compute \( r \times s \)
  
  2) for each attribute \( A \) named in both \( R \) and \( S \), select those tuples from \( r \times s \) where \( r.A = s.A \) (they are called *join attributes*)
  
  3) for each attribute, project out \( s.A \)

- Join selectivity
  
  - if no combination of tuples satisfies the join condition, it results in an empty relation with 0 tuples
  
  - if \( r \) has \( n \) tuples and \( s \) has \( m \) tuples, the result of a join will be between 0 and \( n \times m \) tuples
  
  - join selectivity = expected size of result / \( n \times m \)
  
  - if no join condition to satisfy, join becomes a Cartesian product (also called a *cross join*)
Properties of Join

- Join has a *selecting* power
  - to find $\sigma_{A=a}(r)$,
    define $s(A)$ with a single tuple $t$ such that $t(A)=a$
    and perform $r \times s = \sigma_{A=a}(r)$
  - to find $\sigma_{A=a,B=b}(r)$,
    create a singleton *selection relation* $s$
    with schema $(A, B)$ such that $s = \{ab\}$
    then $r \times s = \sigma_{A=a,B=b}(r)$

- Join is commutative
  $$r \times s = s \times r$$

- Join is associative
  $$r \times (s \times q) = (r \times s) \times q$$
Properties of Join

- Join and projection operators are not inverse of each other, but perform complementary functions

\[ \text{let } q = r \times s \text{ and } r' = \Pi_R(q) \]

then \( \Pi_R(q) = \Pi_R(r \times s) \subseteq r \)

**<ex>**

\[
\begin{array}{c|c|c}
\text{r:} & \text{A} & \text{B} \\
\hline
\text{a1} & \text{b1} \\
\text{a1} & \text{b2} \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{s:} & \text{B} & \text{C} \\
\hline
\text{b1} & \text{c1} \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\text{q:} & \text{A} & \text{B} & \text{C} \\
\hline
\text{a1} & \text{b1} & \text{c1} \\
\end{array}
\]

\[
\Pi_{AB}(q) = r': \begin{array}{c|c}
\text{a1} & \text{b1} \\
\end{array}
\]
Properties of Join

- What if we reverse the order of $\Pi$ and $|\times|$?
  - let $r = \Pi_R(q)$ and $s = \Pi_S(q)$
    where $q$ is a relation with $Q = R \cup S$
    then $q \subseteq r |\times| s = \Pi_R(q) |\times| \Pi_S(q)$
Properties of Join: Example

<table>
<thead>
<tr>
<th>r</th>
<th>A</th>
<th>B</th>
<th>s:</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a1</td>
<td>b1</td>
<td></td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td></td>
<td>a2</td>
<td>b1</td>
<td></td>
<td>b1</td>
<td>c2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r×s:</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td></td>
<td>a1</td>
<td>b1</td>
<td>c2</td>
</tr>
<tr>
<td></td>
<td>a2</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td></td>
<td>a2</td>
<td>b1</td>
<td>c2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>q:</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a1</td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td></td>
<td>a2</td>
<td>b1</td>
<td>c2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>r=Π_R(q):</th>
<th>A</th>
<th>B</th>
<th>s=Π_S(q):</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a1</td>
<td>b1</td>
<td></td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td></td>
<td>a2</td>
<td>b1</td>
<td></td>
<td>b1</td>
<td>c2</td>
</tr>
</tbody>
</table>
Lossless Decomposition

- Relation q decomposes losslessly into schemas R and S if $q = \Pi_R(q) \times \Pi_S(q)$

<ex>

<table>
<thead>
<tr>
<th>r:</th>
<th>A</th>
<th>B</th>
<th>s:</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a1</td>
<td>b1</td>
<td></td>
<td>b1</td>
<td>c1</td>
</tr>
<tr>
<td></td>
<td>a2</td>
<td>b2</td>
<td></td>
<td>b2</td>
<td>c2</td>
</tr>
</tbody>
</table>

$r \times s = q: A B C$

<table>
<thead>
<tr>
<th>r:</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a1</td>
<td>b1</td>
</tr>
<tr>
<td></td>
<td>a2</td>
<td>b2</td>
</tr>
</tbody>
</table>

$r' = \Pi_{AB}(q): A B$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>b1</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>b2</td>
<td></td>
</tr>
</tbody>
</table>


Division

- The relation \( r \div s \) is a relation with schema \( R \rightarrow S \)
such that a tuple \( t \in q = r \div s \), if for every tuple \( T_s \in s \),
there is a tuple \( T_r \in r \) satisfying

\[
\begin{align*}
    t_r(S) &= t_s(S) \\
    t_r(R - S) &= t((R - S)
\end{align*}
\]

- Suitable for queries that include the phrase "for all"

- Division is seldom implemented in practice

\[
\begin{align*}
    r \div s &= \Pi_{R \rightarrow S}^d (r) - \Pi_{R \rightarrow S}^d ((\Pi_{R \rightarrow S} (r) \times s) - r)
\end{align*}
\]
Examples of Division

Find all customers who have an account at all branches located in Brooklyn

Branch-schema (B-name, Assets, B-city)
Deposit-schema (B-name, A#, C-name, Balance)

1) All branches in Brooklyn

$$r_1 = \Pi_{B-name} (\sigma_{B-city}=Brooklyn (\text{branch}))$$

2) All (C-name, B-name) pairs from deposit

$$r_2 = \Pi_{B-name,C-name} (\text{deposit})$$

3) Customers in $$r_2$$ with every branch name in $$r_1$$

$$r_3 = r_2 \div r_1$$

$$= \Pi_{B-name,C-name} (\text{deposit})$$

$$\div \Pi_{B-name} (\sigma_{B-city}=Brooklyn(\text{branch}))$$
Examples of Division

Given the relation showing the faculty-student relationships, find the faculty who teaches every student in the given set.

\[
\begin{array}{cccc}
\text{r: Faculty\#} & \text{Student\#} \\
18 & 105 \\
18 & 106 \\
18 & 108 \\
24 & 105 \\
24 & 106 \\
24 & 107 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{s: Student\#} & \text{s\': Student\#} & \text{s\": Student\#} \\
105 & 106 & 105 \\
108 & 106 & 106 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{r \div s: Faculty\#} & \text{r \div s\': Faculty\#} \\
18 & 18 \\
24 & \\
\end{array}
\]

\[
r \div s" = \emptyset
\]
Renaming

• $\delta_A \leftarrow_B (r)$

• To treat two distinct attributes as identical (for join)

<ex> Class (Name, Birth-date) History (Date, Event)

Who, if any, in the class was born on the day John F. Kennedy was assassinated?

Class $\Join \delta_{B - \text{date}} \leftarrow \text{Date} (\sigma_{\text{Event}=\text{JFK.assassin}} (\text{History}))$

• Two attributes must have the same domain

• Important in practice, but not much used in theory

• Equijoin is similar to join with renaming

Class $\Join \delta_{B - \text{date}} \leftarrow \text{Date} (\text{History}) = (\text{Name, B-date, Event})$