recursion
How expressive is SQL?

• Full programming languages can express all computable functions (C, Java, etc)

Can SQL express all computable queries?

A: YES  B: NO
How expressive is SQL?

Can SQL express the following query: “Is there a way to get from City1 to City2?”

B: NO
“Is there a way to get from City1 to City2 by a direct flight?”

City1 → City2

```sql
select * from flight
where from = 'City1' and to = 'City2'
```
“Is there a way to get from City1 to City2 with at most one stopovers?”

\[
\text{select } * \text{ from flight where from = 'City1' and to = 'City2'}
\]

\[
\text{select x.from, y.to from flight x, flight y where x.from = 'City1' and x.to = y.from and y.to = 'City2'}
\]
“Is there a way to get from City1 to City2 with at most two stopovers?”

\[
\begin{align*}
\text{City1} & \quad \rightarrow \quad \text{City2} \\
\text{OR} & \\
\text{select} & \quad \star \quad \text{from} \quad \text{flight} \\
\text{where} & \quad \text{from} = \text{‘City1’} \quad \text{and} \quad \text{to} = \text{‘City2’} \\
\text{OR} & \\
\text{select} & \quad x.\text{from} \quad , \quad y.\text{to} \\
\text{from} & \quad \text{flight} \ x, \ \text{flight} \ y \\
\text{where} & \quad x.\text{from} = \text{‘City1’} \quad \text{and} \\
& \quad x.\text{to} = y.\text{from} \quad \text{and} \quad y.\text{to} = \text{‘City2’} \\
\text{OR} & \\
\text{select} & \quad x.\text{from} \quad , \quad z.\text{to} \\
\text{from} & \quad \text{flight} \ x, \ \text{flight} \ y, \ \text{flight} \ z \\
\text{where} & \quad x.\text{from} = \text{‘City1’} \quad \text{and} \quad x.\text{to} = y.\text{from} \quad \text{and} \\
& \quad y.\text{to} = z.\text{from} \quad \text{and} \quad z.\text{to} = \text{‘City2’}
\end{align*}
\]
“Is there a way to get from City1 to City2 with at most k stopovers?”

Need $k+1$ tuple variables!
Now

“Is there a way to get from City1 to City2 with any number of stopovers?”

Cannot do in basic SQL!
Similar Examples

• Parts-components relation:
  “Find all subparts of some given part A”
• Parent/child relation:
  “Find all of John’s descendants”
More general: Transitive closure of graph

Find the pairs of nodes \((x, y)\) that are connected by some directed path

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>a</td>
<td>d</td>
</tr>
<tr>
<td>a</td>
<td>e</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
</tr>
<tr>
<td>b</td>
<td>e</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Computing transitive closure $T$ of $G$

“Find the pairs of nodes $<a,b>$ that are connected in $G$”

Same as:
“find pairs of nodes $<a,b>$ at distance 1” UNION
“find pairs of nodes $<a,b>$ at distance at most 2” UNION

.......... 
“find pairs of nodes $<a,b>$ at distance at most $k$” UNION 
.......... 

When to stop?
At some point, no new nodes are added.
Distance cannot be larger than total number of nodes in $G$. 
Example
Example

Distance 1
Example

Distance ≤ 2
Example

Distance $\leq 3$
Algorithm

Denote by $T_k$ the pairs of nodes at distance at most $k$

$T_1$ : “find pairs of nodes <a,b> at distance 1”

\[ \text{select } * \text{ from } G \]

$T_k$ : “find the pairs of nodes <a,b> at distance at most k”

\[ \text{union} \]

\[ \text{(select } * \text{ from } T_{k-1}) \]

\[ \text{where } x.B = y.A) \]
Example

\[ T_1 \]

Diagram:

```
 a---b
  \\
   \ 基
  \\
     c
```

\[ \rightarrow d \rightarrow e \]
Example
Example
Add recursion to SQL

(Not part of the standard)

create recursive view T as
(select * from G)
union
(select x.A, y.B
from G x, T y
where x.B = y.A)

Semantics:
1. Start with empty T
2. While T changes
   {evaluate view with current T;
    union result with T }

Note: This must terminate, since there are finitely many tuples one can add to T (if no new values are created)
One Solution

Add recursion to SQL

Alternative formulation:

with recursive T as
(select * from G)
union
(select x.A, y.B
from G x, T y
where x.B = y.A)
select * from T ;
Another Example

<table>
<thead>
<tr>
<th>frequents</th>
<th>drinker</th>
<th>bar</th>
</tr>
</thead>
</table>

**Friends**: Drinkers who frequent the same bar

Find transitive closure of **Friends**

```sql
create recursive view T as
(select f1.drinker as drinker1, f2.drinker as drinker2
from frequents f1, frequents f2
where f1.bar = f2.bar)
union
(select t1.drinker1, f2.drinker as drinker2
from T t1, frequents f1, frequents f2
where t1.drinker2 = f1.drinker and f1.bar = f2.bar)
```
Problematic example

```sql
create recursive view T as
  (select A, 0 as N from R)
union
  (select A, N+1 as N from T)
```

- Never terminates
- Arithmetic in selects, aggregate functions are forbidden in recursive definitions
Another Solution

Embedded SQL

Powerful way to overcome SQL limitations

Client:
full programming language
(Java, C++, etc)

DB Server

SQL Requests

Answers
Transitive Closure in embedded SQL

\[ T := G \]
\[ \Delta := G \]

while \( \Delta \neq \emptyset \) do

\{ \[ T_{\text{old}} = T \]
\[ T := (\text{select } * \text{ from } T) \]
\[ \text{union} \]
\[ (\text{select } x.A, y.B \text{ from } G x, T y \]
\[ \text{where } x.B = y.A) \]
\[ \Delta := T – T_{\text{old}} \]
\}

Output \( T \)}
Example

$T_1$ and $\Delta_1$
Example

$T_2$ and $\Delta_2$
Example

$T_3$ and $\Delta_3$
Example

\( T_4 = T_3 \) and \( \Delta_4 = \emptyset : \text{Stop!} \)
Algorithm revisited

\[
\begin{align*}
T := G \\
\Delta := G \\
\text{while } \Delta \neq \emptyset \text{ do} \\
\quad \{ \text{T}_{\text{old}} = T \\
\quad T := (\text{select } * \text{ from } T) \\
\quad \text{union} \\
\quad (\text{select } x.A, y.B \text{ from } G x, T y \text{ where } x.B = y.A) \\
\quad \Delta := T - T_{\text{old}} \} \\
\text{Output } T
\end{align*}
\]

Converges in diameter(G) iterations
(maximum distance between two nodes in G)
Optimization: “semi-naïve” evaluation

Use at least one new tuple (from Δ) every time!

\[
\begin{align*}
T &:= G \\
\Delta &:= G \\
\text{while } \Delta \neq \Phi \text{ do} \\
& \quad \{ T_{old} = T \\
& \quad T := (\text{select } \ast \text{ from } T) \\
& \quad \quad \text{union} \\
& \quad \quad (\text{select } x.A, y.B \text{ from } G x, \Delta y \\
& \quad \quad \quad \text{where } x.B = y.A) \\
& \quad \Delta := T - T_{old} \} \\
\text{Output } T
\end{align*}
\]
Example

T_1 and Δ_1
Example

$T_1$ and $\Delta_2$

No longer recomputed $<c,b>$ but recomputed $<c,d>$
Example

$T_1$ and $\Delta_3$

No longer recompute $<a,d>$ but recompute $<c,e>$
Example

$T_4 = T_3$ and $\Delta_4 = \emptyset$: Stop!
Faster Convergence (double recursion)

\[ T := G \]
\[ \Delta := G \]
while \( \Delta \neq \emptyset \) do
\[ \{ T_{\text{old}} = T \]
\[ T := (\text{select } * \text{ from } T) \]
union
(\text{select } x.A, y.B \text{ from } T x, T y \]
where \( x.B = y.A \))
\[ \Delta := T - T_{\text{old}} \]
\}\]
Output \( T \)

Converges in \( \log(\text{diameter}(G)) \) iterations
Example

Focus on computing $\langle a_0, a_8 \rangle$

$\quad a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7 \rightarrow a_8$
Example

Focus on computing $<a_0,a_8>$

\[ a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7 \rightarrow a_8 \]
Example

Focus on computing \( \langle a_0, a_8 \rangle \)
Example

Focus on computing \(<a_0,a_8>\)
Example

Compare to linear recursion (first program)

\[ a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7 \rightarrow a_8 \]
Example

Compare to linear recursion (first program)

\[ a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7 \rightarrow a_8 \]
Example

Compare to linear recursion (first program)

\[ a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7 \rightarrow a_8 \]
Example

Compare to linear recursion (first program)

\[ a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7 \rightarrow a_8 \]
Example

Compare to linear recursion (first program)

\[ a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7 \rightarrow a_8 \]
Example

Compare to linear recursion (first program)

\[ a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7 \rightarrow a_8 \]
Example

Compare to linear recursion (first program)

a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7 \rightarrow a_8
Example

Compare to linear recursion (first program)

\[ a_0 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_4 \rightarrow a_5 \rightarrow a_6 \rightarrow a_7 \rightarrow a_8 \]
Example

Compare to linear recursion (first program)
T := G
Δ := G
while Δ ≠ ∅ do
  \{ T_{old} = T
  T := (select * from T)
  union
  (select x.A, y.B from Δ x, T y
  where x.B = y.A)
  union
  (select x.A, y.B from T x, Δ y
  where x.B = y.A)
  Δ := T − T_{old} \}
Output T
JDBC

- Java Database Connectivity
- Allows SQL to be executed from within Java programs
- Similar to embedded SQL with the following difference:
  - Embedded SQL: SQL processed at compile time
  - JDBC: SQL interpreted at run-time