4 Spatial Queries

4.1 Development of SQL
4.2 Core SQL Queries
4.3 Spatial Queries and Core SQL
4.4 Spatial Extension of the Relational Algebra
4.5 Spatial Extensions of SQL
4.6 Evaluation of Spatial Queries
4.7 Summary
Spatial queries are a special class of queries which are supported by spatial database systems.

Queries are different from queries formulated in "core SQL", offering the use of:

- Geometric data types (point, line, polygon)
- Geometric operations (intersection, length, area)
- Geometric predicates (overlaps, meets, equals)
4.1 Development of SQL

- 1970

- Since 1982
  - SQL/DS, Oracle, DB2
  - Criticism: lack of orthogonality
  - Criticism: lack of functionality

- 1986
  - ANSI SQL
  - Relations, attributes, views
  - SELECT ... FROM ... WHERE ...
4.1 Development of SQL

- 1987
  - SQL86 (ISO/IEC 9075:1986)
- 1989
  - SQL89 (SQL1)
    - ≈ SQL86 + restricted referential integrity
- 1992
  - SQL92 (SQL2)
    - Entry Level
      - ≈ SQL89 + CHECK (attribute)
    - Intermediate Level
      - ⊇ Entry Level + domains, CHECK (relation), CASE, CAST, JOIN, EXCEPT, INTERSECT
4.1 Development of SQL

– Full Level

• ⊇ Intermediate Level + assertions, nested select, nested from

```
SELECT SNO, SNAME, STATUS, CITY, PCT, APCT
FROM S NATURAL JOIN
  (SELECT SNO, COUNT(*) AS PCT
   FROM SP
   GROUP BY SNO) AS XXX
NATURAL JOIN
  (SELECT CITY, AVG(PCT) AS APCT
   FROM S NATURAL JOIN
    (SELECT SNO, COUNT(*) AS PCT
     FROM SP
     GROUP BY SNO) AS YYY
    GROUP BY CITY) AS ZZZ
WHERE PCT > APCT;
```
4.1 Development of SQL

• 1999/2000
  – SQL:1999 (SQL3)
  – ≈ SQL92 + object-orientation, recursive queries, triggers, OLAP, user defined types
  – Computational complete, object-oriented database programming language, descriptive and procedural
  – Core (approx. 180 features)
    • ≈ SQL92 Entry Level + parts of Intermediate and Full Level
  – 9 Packages (approx 240 features)
    • enhanced datetime, enhanced integrity, OLAP, PSM, CLI, basic object support, enhanced object support, trigger, SQL/MM
Example of recursive query (SQL:1999)

```sql
CREATE TABLE FlowsInto (river CHAR(25), riverOrSea CHAR(25), PRIMARY KEY(river));
```

<table>
<thead>
<tr>
<th>river</th>
<th>riverOrSea</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oker</td>
<td>Aller</td>
</tr>
<tr>
<td>Aller</td>
<td>Weser</td>
</tr>
<tr>
<td>Weser</td>
<td>Nordsee</td>
</tr>
<tr>
<td>Elbe</td>
<td>Nordsee</td>
</tr>
<tr>
<td>Isar</td>
<td>Donau</td>
</tr>
<tr>
<td>Lech</td>
<td>Donau</td>
</tr>
<tr>
<td>Inn</td>
<td>Donau</td>
</tr>
<tr>
<td>Donau</td>
<td>Schw. Meer</td>
</tr>
</tbody>
</table>
WITH RECURSIVE FlowsIntoIndirectly(river, riverOrSea) AS
(SELECT river, riverOrSea
 FROM FlowsInto
 WHERE riverOrSea = 'Nordsee'
 UNION
 SELECT FlowsInto.river, FlowsInto.riverOrSea
 FROM FlowsInto AS fi,
     FlowsIntoIndirectly AS fii
 WHERE fi.riverOrSea = fii.river)
SELECT river
FROM FlowsIntoIndirectly;

<table>
<thead>
<tr>
<th>FlowsInto</th>
</tr>
</thead>
<tbody>
<tr>
<td>river</td>
</tr>
<tr>
<td>Oker</td>
</tr>
<tr>
<td>Aller</td>
</tr>
<tr>
<td>Weser</td>
</tr>
<tr>
<td>Elbe</td>
</tr>
<tr>
<td>Isar</td>
</tr>
<tr>
<td>Lech</td>
</tr>
<tr>
<td>Inn</td>
</tr>
<tr>
<td>Donau</td>
</tr>
</tbody>
</table>

\{[[Weser, Nordsee], [Elbe, Nordsee]] \cup
 [[Aller, Weser]] \cup
 [[Oker, Aller]]\}
• Example of type definition (SQL:1999)

```sql
CREATE TYPE address AS (street VARCHAR(35),
city VARCHAR(40),
country CHAR(3));

CREATE TYPE USaddress UNDER address AS (state CHAR(2),
zip ROW (basic INTEGER, plus4 SMALLINT));

METHOD zipcode() RETURNS VARCHAR(10);
```
• Spatial data types

Diagram:

- **ST_Geometry**
  - **ST_Point**
  - **ST_Curve**
    - **ST_CurvePolygon**
  - **ST_Surface**
    - **ST_Polygon**
  - **ST_GeomCollection**
    - **ST_MultiPoint**
    - **ST_MultiSurface**
    - **ST_MultiPolygon**
  - **ST_LineString**
  - **ST_CircularString**
  - **ST_CompoundCurve**
  - **ST_MultiCurve**
  - **ST_MultiLineString**
• Spatial data types
  – ST_Geometry: base type, subtypes are 2D-SDTS
  – ST_Point: point with two coordinates
  – ST_Curve: line, series of points, including interpolated or closed
    – ST_LineString: subtype of ST_Curve, linear interpolation
    – ST_CircularString: subtype of ST_Curve, interpolation by circular arcs
  – ST_Surface: area
  – ST_Polygon: instantiable subtype of ST_Surface with linear rings
Some methods

- `ST_Length()` returns double, length of a curve
- `ST_IsClosed()` returns integer, closed curve?
- `ST_CurveToLine()` returns `ST_LineString`, provides line-approximation of a curve
- `ST_PointN()` returns `ST_Point`, provides nth point of a LineString
- `ST_Area` returns double, provides area of surface
- `ST_Perimeter` returns double, provides perimeter
- `ST_Centroid` returns `ST_Point`, provides centroid
- `ST_ExteriorRing` returns `ST_Curve`, provides outer boundary
4.1 Development of SQL

• 2003
  – SQL:2003
  – Successor of SQL:1999
  – Multi set as an explicit construct (with numerous operations, such as: MULTISET UNION, MULTISET EXCEPT, MULTISET INTERSECT, CARDINALITY)
  – Sequence generators
    • CREATE SEQUENCE <sequence name> AS <type name> [START WITH <value>] [INCREMENT BY <value>] [NO MINVALUE | MINVALUE <value>] [NO MAXVALUE | MAXVALUE <value>] [NO CYCLE | CYCLE]
  – Base type XML for mappings between SQL and XML
4.2 Core SQL Queries

- Simplified syntax of "core Select"

---

Spatial Databases and GIS – Karl Neumann, Sarah Tauscher– Ifis – TU Braunschweig

http://www.ifis.cs.tu-bs.de/sites/default/lecturesMats/ws0809_rdb1/RDB1_08_SQL2.pdf
4.2 Core SQL Queries

• Query 1:
Personnel number, name, amount of remaining leave plus annual leave for all employees, in descending order of amount of vacation days and for the same sum in ascending order by name

Employees (persNo, name, ..., remainingLeave, annualLeave)  
↓
Result (persNo, name, expr.)

SELECT persNo, name, remainingLeave+annualLeave  
FROM Employees  
WHERE remainingLeave IS NOT NULL AND annualLeave IS NOT NULL  
ORDER BY 3 DESC, name ASC;
• Query 2:
  Prerequisites of the course "Programming in Java"
  Course (courseNo, title, ...)
  Prerequisite (courseNo, prerequisiteNo)
  ↓
  Result(title)

SELECT c2.title 
FROM Course c1, Course c2, Prerequisite
WHERE c1.title = 'Programming in Java' AND 
c1.courseNo = Prerequisite.courseNo AND 
Prerequisite.prerequisiteNo = c2.courseNo;
• **Query 3:**

Pairs of employees (personnel number, name), who participate in exactly the same courses

Employees (persNo, name, ...)  Participants (courseNo, persNo)

↓

Result (persNo1, name1, persNo2, name2)

**Question of equality of sets:**

What personnel numbers occur in "Participants" with the same set of course numbers?

Sets A and B are equal:

\[ \forall a \in A: a \in B \land \forall b \in B: b \in A \]
4.2 Core SQL Queries

no universal quantifier in SQL, therefore:
\[
\neg \exists a \in A: a \notin B \land \neg \exists b \in B: b \notin A
\]

```sql
SELECT e1.persNo, e1.name, e2.persNo, e2.name 
FROM Employees e1, Employees e2, Participants p1, Participants p2 
WHERE e1.persNo = p1.persNo 
AND e2.persNo = p2.persNo 
AND p1.persNo < p2.persNo 
AND NOT EXISTS (SELECT * 
    FROM Participants p3 
    WHERE p3.courseNo NOT IN (SELECT p4.courseNo 
        FROM Participants p4 
        WHERE p4.persNo = p2.persNo) 
    AND p1.persNo = p3.persNo) 
AND NOT EXISTS (SELECT * 
    FROM Participants p5 
    WHERE p5.courseNo NOT IN (SELECT p6.courseNo 
        FROM Participants p6 
        WHERE p6.persNo = p1.persNo) 
    AND p2.persNo = p5.persNo) 
ORDER BY e1.persNo, e2.persNo;
```
• Query 4:
Generate a list of potential reviewers for a submitted paper (id = 42). Potential reviewers are only those persons, who are not a coauthor of the respective paper. The potential reviewers should be ordered ascending by the number of the contributions, they already had reviewed.

Person (email, ...)
Review (reviewer, contribution, ...)
Authors (contribution, author)
↓
Result (email, expr.)
4.2 Core SQL Queries

```
SELECT *
FROM ((SELECT email, 0
      FROM Person
      WHERE email NOT IN (SELECT reviewer
                           FROM Review)
      AND email NOT IN (SELECT author
                         FROM Authors
                         WHERE contribution = 42))
      UNION (SELECT reviewer, COUNT(*)
             FROM Review
             WHERE reviewer NOT IN (SELECT author
                                      FROM Authors
                                      WHERE contribution = 42)))
      AS Result
ORDER BY 2;
```
4.3 Spatial Queries and Core SQL

• Core SQL does not provide
  – Geometric data types (point, line, polygon)
  – Geometric operations (intersection, length, area)
  – Geometric predicates (overlaps, meets, equals)

• Have to be relationally modeled and expressed with "core select"

• Works very limited without application programming
• Relational modeling of points
  
  – E.g. id attribute and integer attributes x, y

  ```sql
  CREATE TABLE Points(
    id CHAR(20),
    x INTEGER,
    y INTEGER,
    PRIMARY KEY (id));
  ```

  – Insert points by specifying the appropriate values

  ```sql
  INSERT INTO Points VALUES ( 'restaurant',3540433,5804344);
  INSERT INTO Points VALUES ( 'church',3540441,5804695);
  ```
Distance calculation between points $P_1(x_1, y_1)$, $P_2(x_2, y_2)$

$$\text{distance}(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```sql
SELECT SQRT(((p2.x-p1.x)**2)+((p2.y-p1.y)**2))
FROM Points p1, Points p2
WHERE p1.id = 'restaurant'
AND p2.id = 'church';
```
4.3 Spatial Queries and Core SQL

• Relational modeling of polygons
  – E.g. as list of points
  – Id attribute, list position and integer attributes x, y
  – For each polygon the values of position have to start with value 1 and have to be dense (implementation of a list)

```
CREATE TABLE Polygons ( id CHAR(20),
  position INTEGER,
  x INTEGER,
  y INTEGER,
  PRIMARY KEY (id,position));
```

<table>
<thead>
<tr>
<th>Polygons</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
4.3 Spatial Queries and Core SQL

- N-th point of a polygon (analog: n-th point of a polyline)

```
SELECT p.x, p.y
FROM Polygons p
WHERE p.id = 'P3'
AND p.position = 5;
```

<table>
<thead>
<tr>
<th>Polygons</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>P3</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Spatial Databases and GIS – Karl Neumann, Sarah Tauscher– Ifis – TU Braunschweig
4.3 Spatial Queries and Core SQL

- Smallest rectangle, oriented to the x- and y-axes, that completely encloses the polygon (bounding box); more precisely: the lower left, upper right point of the bounding box

```
SELECT min(p.x), min(p.y), max(p.x), max(p.y)
FROM Polylgons p
WHERE p.id = 'P3'
```
• Application: distance between buildings as distance between the centers of their bounding boxes
4.3 Spatial Queries and Core SQL

```sql
SELECT SQRT(((building1BboxCenterX - building2BboxCenterX)**2) + ((building1BboxCenterY - building2BboxCenterY)**2))
FROM (SELECT (max(p1.X)+min(p1.X))/2 AS building1BboxCenterX,
            (max(p1.Y)+min(p1.Y))/2 AS building1BboxCenterY
     FROM Polygons p1
     WHERE p1.id = 'police'),
     (SELECT (max(p2.X)+min(p2.X))/2 AS building2BboxCenterX,
            (max(p2.Y)+min(p2.Y))/2 AS building2BboxCenterY
     FROM Polygons p2
     WHERE p2.id = 'car park');
```
4.3 Spatial Queries and Core SQL
• Perimeter of polygons

\[ \sum_{i=2}^{n} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} + \sqrt{(x_1 - x_n)^2 + (y_1 - y_n)^2} \]

**Pseudocode solution in embedded SQL**

```sql
INT position, X1, Y1, Xi, Yi, XiMinus1, YiMinus1;
REAL perimeter := 0.0;
BOOL furtherPoints := TRUE;
EXEC SQL DECLARE point CURSOR FOR
SELECT position, x, y FROM Polygons WHERE id = 'P3' ORDER BY position;
```

Spatial Databases and GIS – Karl Neumann, Sarah Tauscher– Ifis – TU Braunschweig 322
4.3 Spatial Queries and Core SQL

EXEC SQL OPEN point;
EXEC SQL FETCH point INTO :position, :X1, :Y1;
XiMinus1 := X1;
YiMinus1 := Y1;
WHILE furtherPoints DO
  BEGIN
    EXEC SQL FETCH point INTO :position, :Xi, :Yi;
    perimeter := perimeter + SQRT((Xi-XiMinus1)^2+(Yi-YiMinus1)^2);
    XiMinus1 := Xi;
    YiMinus1 := Yi;
    IF SQLCODE = 100 THEN furtherPoints := FALSE END;
  END;
perimeter := perimeter + SQRT((X1-XiMinus1)^2+(Y1-YiMinus1)^2);
PRINT perimeter;
EXEC SQL CLOSE point;
4.3 Perimeter of Polygons

• Complexity of queries depends on the chosen spatial data model

• Common data models to represent polygons in relational databases:
  – Topological models (cf. section 2.4)
    • Spaghetti model: polygon as list of points
    • Edge list: polygon as set of edges (unordered)
    • Winged Edge: polygon as set of edges (ordered)
4.3 Perimeter of Polygons

- Spaghetti model, one polygon

```
SELECT uid,
       pm1 + (SQRT(((p.x – pn.x)**2) +((p.y – pn.y)**2))) AS perimeter
FROM polygons p, polygons pn,
     (SELECT p1.id AS uid,
          SUM(SQRT(((p2.x – p1.x)**2) +((p2.y – p1.y)**2))) AS pm1
     FROM Polygons p1, Polygons p2
     WHERE p1.id = p2.id
     AND p1.position = p2.position - 1
     AND p1.id = ‘P3’)
WHERE p.id = pn.id
AND p.id = uid
AND p.position = 1
AND pn.position = (SELECT max(m.position)
                      FROM Polygons m
                      WHERE m.id = pn.id)
```

<table>
<thead>
<tr>
<th>Polygons</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>pos</td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>x1</td>
<td>y1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>x2</td>
<td>y2</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>x3</td>
<td>y3</td>
</tr>
<tr>
<td>P3</td>
<td>4</td>
<td>x4</td>
<td>y4</td>
</tr>
<tr>
<td>P3</td>
<td>5</td>
<td>x5</td>
<td>y5</td>
</tr>
<tr>
<td>P3</td>
<td>6</td>
<td>x6</td>
<td>y6</td>
</tr>
<tr>
<td>P3</td>
<td>7</td>
<td>x7</td>
<td>y7</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.3 Perimeter of Polygons

- Spaghetti model, all polygons

```
SELECT uid,
    pm1 + (SQRT(((p.x - pn.x)**2) 
    +((p.y - pn.y)**2))) AS perimeter
FROM polygons p, polygons pn,
(SELECT p1.id AS uid,
    SUM(SQRT(((p2.x - p1.x)**2) 
    +((p2.y - p1.y)**2))) AS pm1
FROM Polygons p1, Polygons p2
WHERE p1.id = p2.id
AND p1.position = p2.position - 1
GROUP BY p1.id )
WHERE p.id = pn.id
AND p.id = uid
AND p.position = 1
AND pn.position = (SELECT max(m.position)
FROM Polygons m
WHERE m.id = pn.id)
```

```
<table>
<thead>
<tr>
<th>id</th>
<th>position</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>x1</td>
<td>y1</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>x2</td>
<td>y2</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>x3</td>
<td>y3</td>
</tr>
<tr>
<td>P3</td>
<td>4</td>
<td>x4</td>
<td>y4</td>
</tr>
<tr>
<td>P3</td>
<td>5</td>
<td>x5</td>
<td>y5</td>
</tr>
<tr>
<td>P3</td>
<td>6</td>
<td>x6</td>
<td>y6</td>
</tr>
<tr>
<td>P3</td>
<td>7</td>
<td>x7</td>
<td>y7</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
4.3 Perimeter of Polygons

• Edge list, one polygon

```sql
SELECT SUM(SQRT(((end.x - start.x)**2) + ((end.y - start.y)**2))) AS perimeter
FROM Edge e, Node start, Node end
WHERE e.startnode = start.id
AND e.endnode = end.id
AND (e.rightface = 'P3'
OR e.leftface = 'P3')
```

<table>
<thead>
<tr>
<th>Node</th>
<th>id</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>x1</td>
<td>y1</td>
<td></td>
</tr>
<tr>
<td>V2</td>
<td>x2</td>
<td>y2</td>
<td></td>
</tr>
<tr>
<td>V3</td>
<td>x3</td>
<td>y3</td>
<td></td>
</tr>
<tr>
<td>V4</td>
<td>x4</td>
<td>y4</td>
<td></td>
</tr>
<tr>
<td>V5</td>
<td>x5</td>
<td>y5</td>
<td></td>
</tr>
<tr>
<td>V6</td>
<td>x6</td>
<td>y6</td>
<td></td>
</tr>
<tr>
<td>V7</td>
<td>x7</td>
<td>y7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Edge</th>
<th>id</th>
<th>start node</th>
<th>end node</th>
<th>right face</th>
<th>left face</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>V1</td>
<td>V2</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>V2</td>
<td>V3</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>V3</td>
<td>V4</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td>V4</td>
<td>V5</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td>V5</td>
<td>V6</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E6</td>
<td>V6</td>
<td>V7</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E7</td>
<td>V7</td>
<td>V1</td>
<td>P3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.3 Perimeter of Polygons

- **Edge list, all polygons**

```sql
SELECT l.polygon, pmR + pmL AS perimeter
FROM (SELECT SUM(SQRT(((end.x – start.x)**2) +((end.y – start.y)**2))) AS pmR
    rightface AS polygon
    FROM edge e, Node start, Node end
    WHERE e.startnode = start.id
    AND e.endnode = end.id
    GROUP BY rightface) AS r
FULL JOIN (SELECT SUM(SQRT(((end.x – start.x)**2) +((end.y – start.y)**2))) AS pmL
    leftface AS polygon
    FROM edge e, Node start, Node end
    WHERE e.startnode = start.id
    AND e.endnode = end.id
    GROUP BY leftface) AS l
ON l.polygon = r.polygon
```

<table>
<thead>
<tr>
<th>id</th>
<th>start node</th>
<th>end node</th>
<th>right face</th>
<th>left face</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>V1</td>
<td>V2</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>V2</td>
<td>V3</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>V3</td>
<td>V4</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td>V4</td>
<td>V5</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td>V5</td>
<td>V6</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>E6</td>
<td>V6</td>
<td>V7</td>
<td>P3</td>
<td></td>
</tr>
<tr>
<td>E7</td>
<td>V7</td>
<td>V1</td>
<td>P3</td>
<td></td>
</tr>
</tbody>
</table>
4.3 Perimeter of Polygons

- **Winged edge model: Same query**

<table>
<thead>
<tr>
<th>id</th>
<th>start node</th>
<th>end node</th>
<th>right face</th>
<th>left face</th>
<th>left arm</th>
<th>right arm</th>
<th>left leg</th>
<th>right leg</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>V1</td>
<td>V2</td>
<td>P3</td>
<td></td>
<td>E2</td>
<td>E7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td>V2</td>
<td>V3</td>
<td>P3</td>
<td></td>
<td>E3</td>
<td>E1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>V3</td>
<td>V4</td>
<td>P3</td>
<td>E2</td>
<td>E4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Determine the edges of the polygon/face p in the correct order**
  
  - Winged edge model, clockwise:
    - p is right face: the next edge is the right arm
    - p is left face: the next edge is the left leg
4.3 Perimeter of Polygons

- Edge list, clockwise:
  - $p$ is right face: one node of the next edge has the same coordinates as the end node of the current edge, and $p$ is its left or right face
  - $p$ is left face: one node of the next edge has the same coordinates as the start node of the current node, and $p$ is its left or right face

- Both impossible with „Core SQL”

- Trivial in the spaghetti model

SELECT $x, y$
FROM Polygon
WHERE id = ‘P3’
ORDER BY position
• Relational algebra
  – Formal language for formulating queries on relations
  – A query language is relationally complete, if each term of the relation algebra can be formulated with the query language
  – If each term can be formulated by exactly one single query, the query language is strongly complete
  – Operations are on (base) relations and deliver relations (closed)
  – There are base operations and derived operations
4.4 Spatial Extensions of the Relational Algebra

- **Base operations**
  - Projection: \( \pi_{A_1,A_2,\ldots,A_k}(R) \)
    - \( \pi_{\text{persNo},\text{name}}(\text{Employees}) \)
  - Selection: \( \sigma_\phi(R) \)
    - \( \sigma_{\text{remainingLeave}>10}(\text{Employees}) \)
  - Cross product: \( R \times S \)
    - Course \( \times \) Prerequisite
  - Union: \( R \cup S \)
    - \( \pi_{\text{email}}(\text{Person}) \cup \pi_{\text{email}}(\text{Institute}) \)
  - Difference: \( R \setminus S \)
    - \( \pi_{\text{persNo}}(\text{Employees}) \setminus \pi_{\text{persNo}}(\text{Participants}) \)
  - Rename: \( \delta_{A \leftarrow B}(R) \)
    - \( \delta_{\text{name} \leftarrow \text{surname}}(\text{Employees}) \)
• **Derived operations**

  - **Intersection**
    
    \[
    R \cap S = \pi_{\text{reviewer}}(\text{Review}) \cap \pi_{\text{author}}(\text{Authors})
    \]

  - **Join**
    
    \[
    R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)
    \]

    **Example**
    
    \[
    \text{Review} \bowtie_{\text{reviewer} \neq \text{author}} \text{Authors}
    \]

  - **Natural join**
    
    \[
    R \bowtie S = \pi_{A_1,\ldots,A_k,C_1,\ldots,C_m,B_1,\ldots,B_n} \\
    \left(\sigma_{R.C_1=S.C_1 \land \ldots \land R.C_m=S.C_m}(R \times S)\right)
    \]

    **Example**
    
    \[
    \text{Course} \bowtie \text{Prerequisite}
    \]
Example (Query 2):
Prerequisites of the course "Programming in Java"

Course (courseNo, title, ...)
Prerequisite (courseNo, prerequisiteNo)

\[
\downarrow
\]

Result (title)

(1) \[ \pi_{title_2}(\sigma_{title_1=’Programming in Java’}(\sigma_{courseNo_1=courseNo_3}(\sigma_{courseNo_2=prerequisiteNo}(Course \times Course \times Prerequisite)))) \]

(2) \[ \pi_{title}(\sigma_{courseNo=prerequisiteNo}(Course\times \pi_{prerequisiteNo}(\sigma_{courseNo_1=courseNo_2}(Prerequisite\times \pi_{courseNo}(\sigma_{title=’Programming in Java’}(Course))))))) \]
4.4 Spatial Extensions of the Relational Algebra

- Solution 2 as operator tree

```
π

σ_{\text{courseNo} = \text{prerequisiteNo}}

\times

π_{\text{prerequisiteNo}}

σ_{\text{courseNo}_1 = \text{courseNo}_2}

\times

π_{\text{courseNo}}

σ_{\text{title} = \text{Prgr... Java'}}

\times

Course

π_{\text{title}}
```

Spatial Databases and GIS – Karl Neumann, Sarah Tauscher– Ifis – TU Braunschweig
4.4 Spatial Extensions of the Relational Algebra

- As data types of the attributes the “usual data types” are presupposed implicitly
  - Integer
  - Decimal
  - String
  with the usual operators such as +, −, *, / and predicates as <, =, >, ≠

- Operators and predicates can only be used in the qualification formulas of the selection, e.g.:
  \[ \sigma_{\text{remainingLeave}>10}(\text{Employees}) \]
  \[ \sigma_{\text{remainingLeave}<0.2*\text{annualLeave}}(\text{Employees}) \]
4.4 Spatial Extensions of the Relational Algebra

• Query results are always tuples of attribute values which are unchanged (no compositions of values by operators)

• For example, query 1' can not be formulated:
  Personnel number, name, amount of remaining leave plus annual leave for all employees

  Employees (persNo, name, ..., remainingLeave, annualLeave)

  ↓

  Result (persNo, name, expr.)

  \[ \Pi_{\text{persNo}, \text{name}, \text{remainingLeave} + \text{annualLeave}} (\text{Employees}) \]
4.4 Spatial Extensions of the Relational Algebra

- Extensions typically on the level of data types
- Thus, introducing new sorts, operators, predicates (e.g., analogous to SQL/MM Part 3: Spatial)
  - Point, LineString, Polygon
  - Length, Area, Perimeter, Centroid
  - Equals, Disjoint, Intersects, Overlaps, Touches, Contains
4.4 Spatial Extensions of the Relational Algebra

• Attributes now may also be based on spatial data types, e.g.:
  
  – Building (id: String, typeOfUse: String, groundPlan: Polygon)
  
  – Parcel (id, groundPlan: Polygon)
  
  – Person (persNr, name, firstName, ...)
  
  – Owner(persNo, buildingId)

• Now, spatial operators and predicates may be used in the qualification formulas of the selection
• **Query 5:**

Ids of all buildings on the parcel with id "1222"

Building(id: String, typeOfUse: String, groundPlan: Polygon)
Parcel (id, groundPlan: Polygon)

\[\prod_{id_2} (\sigma_{\text{Contains}(\text{groundPlan}_1, \text{groundPlan}_2)} (\sigma_{id_1='1222'} (\text{Parcel} \times \text{Building})))\]
• Query 6:
  
  Ids of churches which are located no more than 2 km from a car park

\[ \pi_{id} \left( \sigma_{\text{Distance}(\text{Centroid}(\text{groundPlan}_1), \text{Centroid}(\text{groundPlan}_2)) < 2000} \left( \sigma_{\text{typeOfUse}_1 = \text{'church'} \land \text{typeOfUse}_2 = \text{'car park'}} \right) \right) \]

\[ (\text{Building} \times \text{Building}) \]
4.4 Spatial Extensions of the Relational Algebra

• Query 7:

Names of owners of adjacent residential buildings

Building(id: String, typeOfUse: String, groundPlan: Polygon)
Person(persNo, name, firstName, ...)
Owner(persNo, buildingId)

\[
\pi_{\text{name}_1, \text{name}_2} \\
(\sigma_{\text{persNo}_1 = \text{persNo}_3 \land \text{persNo}_2 = \text{persNo}_4} \\
(\text{Person} \times \text{Person}) \\
(\sigma_{\text{buildingId}_1 = \text{id}_1 \land \text{buildingId}_2 = \text{id}_2} \\
(\text{Owner} \times \text{Owner}) \\
(\pi_{\text{id}_1, \text{id}_2} (\sigma_{\text{Touches}(\text{groundPlan}_1, \text{groundPlan}_2)} \\
(\sigma_{\text{id}_1 < \text{id}_2} (\sigma_{\text{typeOfUse}_1 = 'residential' \land \text{typeOfUse}_2 = 'residential')} \\
(\text{Building} \times \text{Building})))))))))))
\]
• Query 5’:
Areas of all buildings on the parcel with id "1222"

Building(id: String, typeOfUse: String, groundPlan: Polygon)
Parcel (id, groundPlan: Polygon)

\[ \pi_{\text{Area}(\text{groundPlan}_2)}(\sigma_{\text{Contains}(\text{groundPlan}_1,\text{groundPlan}_2)}(\sigma_{\text{id}_1='1222'}(\text{Parcel} \times \text{Building}))) \]

• As in the "classical" relational algebra no operators are allowed in the projection list
4.5 Spatial Extensions of SQL

- Extensions on the level of data types (again)
- Thus, new sorts, operators, predicates
- Typically orientation at standard
- SQL/MM Part 3: Spatial, e.g.:
  - Point, LineString, Polygon
  - Length, Area, Perimeter, Centroid
  - Equals, Disjoint, Intersects, Overlaps, Touches, Contains
- Relations now may have spatial attributes
4.5 Spatial Extensions of SQL

- Use of spatial data types in the same way as the "usual" data types in CREATE TABLE statement, e.g.:

  CREATE TABLE Building
  (id CHAR(25),
  typeOfUse CHAR(25),
  groundPlan Polygon,
  PRIMARY KEY(id));

  CREATE TABLE Parcel
  (id CHAR(25),
  groundPlan Polygon,
  PRIMARY KEY(id));

  INSERT INTO Building VALUES ( 'G4211', 'police', (12125, 1333, 13430, 1560, 13260, 2497, 14111, 2695, 14111, 2638, 16040, 3092, 15303, 6468, 13345, 5958, 13771, 3943, 12948, 3773, 12948, 3887, 11671, 3631));
4.5 Spatial Extensions of SQL

- Spatial operators and predicates now may occur in SELECT statements, at the same places where the "usual" operators and predicates are allowed.
4.5 Spatial Extensions of SQL

Query 5:
Ids of all buildings on the parcel with id "1222"

```
SELECT b.id
FROM Building b,
     Parcel p
WHERE p.id = '1222' AND
      Contains(p.groundPlan, b.groundPlan);
```
**Query 6:**

Ids of churches which are located no more than 2 km from a car park

```sql
SELECT b1.id
FROM Building b1, Building b2
WHERE b1.typeOfUse = 'church'
AND b2.typeOfUse = 'car park'
AND Distance(Centroid(b1.groundPlan), Centroid(b2.groundPlan)) < 2000;
```
4.5 Spatial Extensions of SQL

• Query 7:
  Names of owners of adjacent residential buildings

```sql
SELECT p1.name, p2.name
FROM Person p1, Person p2, Building b1, Building b2, Owner o1, Owner o2
WHERE p1.persNo = o1.persNo AND p2.persNo = o2.persNo
AND b2.typeOfUse = 'residential' AND o2.buildingId = b2.id
AND b1.typeOfUse = 'residential' AND o1.buildingId = b1.id
AND Touches(b1.groundPlan,b2.groundPlan)
AND b1.id < b2.id;
```
4.5 Spatial Extensions of SQL

• Query 5’:

Areas of all buildings on the parcel with id "1222"

```
SELECT Area(b.groundPlan)
FROM Building b, Parcel p
WHERE p.id = '1222'
AND Contains(p.groundPlan, b.groundPlan);
```
4.5 Spatial Extensions of SQL

• Query 8:

What percentage of the area of the (planned) route '647' is covered by the parcel '586'?

```
SELECT (Area (Intersection (r.geometry,p.groundPlan)) / Area(r.geometry)) * 100
FROM Route r, Parcel p
WHERE r.id = '647'
AND p.id = '586';
```
• Query 9:
Through which parcels do brooks flow?

Parcel (id, groundPlan: Polygon)
Brook(...,geometry: Polygon)

SELECT p.id
FROM Parcel p, Brook b
WHERE Intersects (p.groundPlan, b.geometry);
4.5 Spatial Extensions of SQL

• Typical spatial SQL extensions are provided by
  – Oracle Spatial
  – Postgres

• Oracle Spatial
  – Extension of the Oracle Database System
  – A (very special) geometric data type:
    ```sql
    CREATE TYPE sdo_geometry AS OBJECT ( 
    SDO_GTYPE NUMBER, 
    SDO_SRID NUMBER, 
    SDO_POINT SDO_POINT_TYPE, 
    SDO_ELEM_INFO MDSYS.SDO_ELEM_INFO_ARRAY, 
    SDO_ORDINATES MDSYS.SDO_ORDINATE_ARRAY);
    ```
4.5 Spatial Extensions of SQL

• By SDO_GTYPE the geometry type is coded
  – 0 UNKNOWN_GEOMETRY
  – 1 POINT
  – 2 LINESTRING
  – 3 POLYGON
  – 4 Collection
  – 5 MULTIPOINT
  – 6 MULTILINESTRING
  – 7 MULTIPOLYGON
4.5 Spatial Extensions of SQL

- SDO_SRID will specify the spatial reference system (currently not used)
- SDO_POINT_TYPE represents three-dimensional point (currently only two dimensions are supported)
- SDO_POINT is only used if one point is present and the following two attributes are NULL (efficient coding of individual points)
- SDO_ELEM_INFO array of numbers indicates how the values in
- SDO_ORDINATES are to be interpreted
4.5 Spatial Extensions of SQL

• Example: relation for buildings

```
CREATE TABLE Building
(id CHAR(25),
typeOfUse CHAR(25),
groundPlan SDO_GEOMETRY,
PRIMARY KEY(id));
```

```
INSERT INTO Building VALUES ('G4211','police',
SDO_GEOMETRY(SDO_GTYPE = 2003,
SDO_SRID = NULL, SDO_POINT = NULL,
SDO_ELEM_INFO = (1,1003,1),
SDO_ORDINATES = (12125,1333,13430,1560,
13260,2497,14111,2695,14111,2638,16040,
13092,15303,6468,13345,5958,13771,3943,
12948,3773,12948,3887,11671,3631)));
```
• Polygon with hole
  – Distinction between inner and outer polygon rings
  – Connection of points as straight line or circular arc

```
SDO_GEOMETRY ( 2003, NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,1, 19,2003,1), SDO_ORDINATE_ARRAY(2,4, 4,3, 10,3, 13,5, 13,9, 11,13, 5,13, 2,11, 2,4, 7,5, 7,10, 10,10, 10,5, 7,5) )
```

http://download.oracle.com/docs/...5/sdo_objrelschemah.htm
• Polygon with hole
  – Distinction between inner and outer polygon rings
  – Connection of points as straight line or circular arc

SDO_GEOMETRY
( 2003, 2-dimensional polygon
NULL,
NULL,
SDO_ELEM_INFO_ARRAY(1,1003,1, 19,2003,1),
SDO_ORDINATE_ARRAY(2,4, 4,3, 10,3, 13,5, 13,9, 11,13, 5,13, 2,11, 2,4,7,5, 7,10, 10,10, 10,5, 7,5) )
• Compound line
  – Made up of straight line segments and circular arcs
  – Arcs are described by three coordinate pairs

```
SDO_GEOMETRY(2002, NULL, NULL,
  SDO_ELEM_INFO_ARRAY(1,4,2, 1,2,1, 3,2,2),
  SDO_ORDINATE_ARRAY(10,10, 10,14, 6,10, 14,10) )
```
• Compound line
  – Made up of straight line segments and circular arcs
  – Arcs are described by three coordinate pairs

SDO_GEOMETRY(
  2002, NULL,
  SDO_ELEM_INFO_ARRAY(1,4,2, 1,2,1, 3,2,2),
  SDO_ORDINATE_ARRAY(10,10, 10,14, 6,10, 14,10)
)
• Collection
  – Contains an arbitrary number of geometries of different types

SDO_GEOMETRY (
  2004,
  NULL,
  NULL,
  SDO_ELEM_INFO_ARRAY (1,1,1, 3,2,1, 7,1003,1),
  SDO_ORDINATE_ARRAY (12,14, 2,3, 10,7, 5,8, 9,8, 10,13, 5,13, 5,8))
• Query 5’:

Areas of all buildings on the parcel with id "1222"

```sql
SELECT SDO_GEOM.SDO_Area(b.groundPlan, 0.005) FROM Building b, Parcel p WHERE p.Id = '1222' AND SDO_CONTAINS(p.groundPlan, b.groundPlan) = 'TRUE'
```
• Query 5’:
Areas of all buildings on the parcel with id "1222"

```
SELECT SDO_GEOM.SDO_Area(b.groundPlan, 0.005) FROM Building b, Parcel p
WHERE p.Id = '1222' AND SDO_RELATE(p.groundPlan, b.groundPlan, 'mask=contains') = 'TRUE'
```
4.5 Spatial Extensions of SQL

• Postgres
  – Open source software database system with spatial extensions

• Data types
  – point (float, float)
  – box (point, point)
4.5 Spatial Extensions of SQL

- \( \text{lseg} \)  
  \((\text{point}, \text{point})\)

- \( \text{line} \)  
  \((\text{point}_1, \text{point}_2, ..., \text{point}_n)\)  
  \(2 \leq n \leq 124\)

- \( \text{long line} \)  
  \((\text{point}_1, \text{point}_2, ..., \text{point}_n)\)  
  \(2 \leq n \leq 100.000.000\)
4.5 Spatial Extensions of SQL

- polygon
  \((point_1, point_2, ..., point_n)\)
  \(3 \leq n \leq 124\)

- long polygon
  \((point_1, point_2, ..., point_n)\)
  \(3 \leq n \leq 100,000,000\)

- circle
  \((point, float)\)
4.5 Spatial Extensions of SQL

• Predicates
  – =, <> point×point, lseg×lseg, ..., circle×circle
  – intersects
    spatial_type×spatial_type
  – inside
    point×{box, polygon, long polygon, circle}
    box×{box, polygon, long polygon, circle}
    lseg×{box, polygon, long polygon, circle}
    line×{box, polygon, long polygon, circle}
    long line×{box, polygon, long polygon, circle}
    polygon×{box, polygon, long polygon, circle}
    long polygon×{box, polygon, long polygon, circle}
    circle×{box, polygon, long polygon, circle}
4.5 Spatial Extensions of SQL

- Functions
  - Area
    \{box, circle, polygon, long polygon\} → float
  - Length
    \{lseg, line, long line\} → float
  - Perimeter
    \{box, circle, polygon, long polygon\} → float
  - Distance 
    point × point → float
  - Point_x, Point_y
    point → float
  - Box_ll, Box_ur
    box → point
  - bbox
    \{box, lseg, line, long line, polygon, long polygon, circle\} → box
4.5 Spatial Extensions of SQL

• Example: relation for buildings

CREATE TABLE Building
(id CHAR(25),
typeOfUse CHAR(25),
groundPlan Polygon(50),
PRIMARY KEY(id));

INSERT INTO Building VALUES ( 'G4211', 'police',
((12125,1333),(13430,1560),(13260,2497),
(14111,2695),(14111,2638),(16040,3092),
(15303,6468),(13345,5958),(13771,3943),
(12948,3773),(12948,3887),(11671,3631)) );
• Query 5’:
Areas of all buildings on the parcel with id "1222"

```
SELECT Area(b.groundPlan) FROM Building b, Parcel p WHERE p.id = '1222' AND inside(b.groundPlan,p.groundPlan);
```
4.5 Spatial Extensions of SQL

- Distances between car parks and police stations
SELECT b1.id, b2.id, 
Distance ( 
point ( (Point_X (box_ll (bbox (b1.groundPlan)))) + 
Point_X (box_ur (bbox (b1.groundPlan)))) / 2, 
(Point_Y (box_ll (bbox (b1.groundPlan)))) + 
Point_Y (box_ur (bbox (b1.groundPlan)))) / 2), 
point ( (Point_X (box_ll (bbox (b2.groundPlan)))) + 
Point_X (box_ur (bbox (b2.groundPlan)))) / 2, 
(Point_Y (box_ll (bbox (b2.groundPlan)))) + 
Point_Y (box_ur (bbox (b2.groundPlan)))) / 2)) 
FROM Building b1, Building b2 
WHERE b1.typeOfUse = 'car park' AND 
b2.typeOfUse = 'police' 
ORDER BY 1, 3;
Spatial queries are frequently classified by

- **Point query**
  given: object o with point geometry
  find: \( \{ o_i \mid \text{contains}(o_i, \text{geometry}, o.\text{geometry}) \} \)

- **Window query**
  given: object o with rectangle geometry
  find: \( \{ o_i \mid \text{intersects}(o_i, \text{geometry}, o.\text{geometry}) \} \)

- **Region query**
  given: object o with polygon geometry
  find: \( \{ o_i \mid \text{intersects}(o_i, \text{geometry}, o.\text{geometry}) \} \)
4.6 Evaluation of Spatial Queries

- Spatial join
given: object classes o, o'
find: \{ o_i \in o, o_j \in o' \mid \theta(o_i.geometry, o_j.geometry)\}
with \theta : =, <>, intersects, inside

- Examples:
Query 5':
Areas of all buildings on the parcel with id "1222"

```
SELECT Area(b.groundPlan)
FROM Building b, Parcel p
WHERE p.id = '1222'
     AND inside(b.groundPlan, p.groundPlan)
```
Query 9:
Through which parcels do brooks flow?

```sql
SELECT p.id
FROM Parcel p,
     Brook b
WHERE Intersects (p.groundPlan, b.geometry);
```
4.6 Evaluation of Spatial Queries

- Naive evaluation of spatial joins (nested loop join) too inefficient:
  
given: object classes o, o’
find: \{ o_i \in o, o_j \in o’ | \theta(o_i.\text{geometry}, o_j.\text{geometry}) \}

hits := \emptyset
for all o_i \in o do
begin
  for all o_j \in o’ do
  begin
    if \theta(o_i.\text{geometry}, o_j.\text{geometry})
    then hits := hits \cup [o_i, o_j] fi
  end
end

Spatial Databases and GIS – Karl Neumann, Sarah Tauscher– Ifis – TU Braunschweig
4.6 Evaluation of Spatial Queries

- Evaluating query 9 (through which parcels do brooks flow?) with e.g. 5,000 polygons of parcels and 40 polygons of brooks results in 200,000 intersection operations, e.g.:

```
intersects((7200,2430,7245,2430,7200,2565,7245,2700,7290,2790,7335,7245,3870,7110,3780,6975,3870,6930,3960,6840,4005,6750,4095,6570,4230,6480,4230,6480,4185,6705,4050,6840,3915,7020,3780,7110,3735,7245,3780,7335,3780,7425,3645,7470,3375,7515,3195,7335,2970,7200,7245,7155,2520,7200,2385,7245,2385,7200,2430),(6975,2565,7065,2835,7245,2835,7875,2790,8190,2970,8010,3195,7650,3555,7470,3510,7470,3690,7290,3825,7020,3825,6750,4050,6570,4095,6480,3915,6345,3960,6210,3600,6165,3420,6165,3240,6210,3015,6300,2655,6345,2610,6840,2610,6975,2565))
```
• Therefore evaluation of spatial queries in 2 steps
  – Filter step
    • Determination of possible hits by evaluation on spatial approximation (lower costs)
  – Refinement step
    • Evaluation on accurate geometry only for objects of the filter step
4.6 Evaluation of Spatial Queries

- Conservative approximation
  - Complete coverage of the object
  - Recognition of false hits
    \[ \neg (\theta(\text{ConAppr}(o_i.\text{geometry}), \text{ConAppr}(o_j.\text{geometry}))) \]
    \[ \Rightarrow \]
    \[ \neg (\theta(o_i.\text{geometry}, o_j.\text{geometry})) \]
4.6 Evaluation of Spatial Queries

- Progressive approximation
  - Completely covered by the object
  - Recognition of hits
    \[ \theta(\text{ProgAppr}(o_i.\text{geometry}), \text{ProgAppr}(o_j.\text{geometry})) \]
    \[ \Rightarrow \]
    \[ \theta(o_i.\text{geometry}, o_j.\text{geometry}) \]
  - Complex computation

http://dbs.mathematik.uni-marburg.de/teaching
In practice usually conservative approximation by minimum bounding (paraxial) rectangle (MBR, bounding box)
4.6 Example (Query 9)

• Initial situation
4.6 Example (Query 9)

• Some parcels and brooks are marked
4.6 Example (Query 9)

- MBRs of parcels and brooks
4.6 Example (Query 9)

- Overlapping MBRs
4.6 Example (Query 9)

- Overlapping parcels and brooks
Evaluation of spatial joins with MBRs (somewhat) more efficient:

\[
\text{hits} := \emptyset \\
\text{for all } o_i \in o \text{ do begin}
    \text{for all } o_j \in o' \text{ do begin}
        \text{if } \theta(\text{MBR}(o_i\text{-geometry}), \text{MBR}(o_j\text{-geometry}))
            \text{then if } \theta(o_i\text{-geometry}, o_j\text{-geometry})
                \text{then hits} := \text{hits} \cup [o_i, o_j]
        \text{fi}
    \text{fi}
\text{end end}
\]
4.6 Evaluation of Spatial Queries

- There are still $|o_i| \cdot |o_j|$ comparisons needed (e.g. 200,000 comparisons between 5,000 MBRs of parcels and 40 MBRs of brooks)
- For a more efficient evaluation of spatial queries, spatial indexes are needed.

http://www.geoinformation.net/
4.7 Summary

- Spatial queries are offering the use of
  - Geometric data types
  - Geometric operations
  - Geometric predicates

- Development of SQL
  - SQL92 (SQL2), SQL:1999 (SQL3)
  - SQL/MM Part 3: Spatial
  - Data types, methods

- Core SQL queries
4.7 Summary

- Spatial queries and core SQL
  - Relational modeling of points
  - Distance calculation between points
  - Relational modeling of polygons
  - N-th point of a polygon
  - Bounding box
  - Perimeter of polygons
  - Detour [perimeter of polygons, direct]
4.7 Summary

- Spatial extensions of the relational algebra
  - Base operations, derived operations
  - Extensions by new sorts, operators, predicates
  - Some queries

- Spatial extensions of SQL
  - Some queries
  - Oracle Spatial
  - Detour [SDO_Geometry]
  - Postgres
4.7 Summary

• Evaluation of spatial queries
  – Point query
  – Window query
  – Region query
  – Spatial join
  – Evaluation of spatial queries in 2 steps
  – Conservative approximation
  – Progressive approximation
4.7 Summary

- GIS: collect, manage, analyse, display
- Evaluation of queries
- Approximations
- Spatial extensions
- Spatial Databases and GIS – Karl Neumann, Sarah Tauscher – Ifis – TU Braunschweig
- SQL
- Relational algebra
- Queries in RDBs