Advanced SQL - Subqueries and Complex Joins

Outline for Today:

- The URISA Proceedings database - more practice with increasingly complicated SQL queries
- Advanced Queries:
  - **Sub-queries**: one way to nest or a cascade query is to stick a query in the 'where' clause: e.g., find parcels owned by XXX from that set of parcels that had a fire. This is a powerful way to take advantage of the fact that any SQL query returns a table - which can they be the starting point of another SQL query.
  - **Self-joins**: the 'where' clause can become quite complex with many joins and related 'and' and 'or' conditions. But handling 'and' conditions is a little tricky. How can you find papers the use **both** keyword Y and keyword Z if the table relating papers and keywords only shows one pair at a time?
- The zoning variance database
  - Understanding the schema and rationale for the Boston zoning variance database (which we use later to map them as study spatial patterns as well as to illustrate concepts about distributed databases and community empowerment.
  - Using the history of zoning database to understand how real databases evolve over time

More URISA database Queries

- ...from the URISA database* page
- Additional notes on SQL*Plus formatting* added to SQL Notes*

Advanced Queries: Subqueries

A subquery can be nested within a query

* Kindly refer to Lecture Notes section
Example: Find the parcel with the highest estimated loss from a fire

```sql
SELECT *
FROM FIRES
WHERE ESTLOSS =
    (SELECT MAX(ESTLOSS)
     FROM FIRES);
```

Alternatively, include the subquery as an inline "table" in the FROM clause:

```sql
SELECT F.*
FROM FIRES F,
    (SELECT MAX(ESTLOSS) MAXLOSS
     FROM FIRES) M
WHERE F.ESTLOSS = M.MAXLOSS;
```

Example: Find the parcels that have not had a fire

```sql
SELECT *
FROM PARCELS
WHERE PARCELID NOT IN
    (SELECT PARCELID
     FROM FIRES);
```

or, more efficiently,

```sql
SELECT *
FROM PARCELS P
WHERE NOT EXISTS
    (SELECT NULL
     FROM FIRES F
     WHERE P.PARCELID = F.PARCELID);
```

Example: Find the parcels that have not obtained a permit:

```sql
SELECT *
FROM PARCELS
WHERE (PID, WPB) NOT IN
    (SELECT PID, WPB
     FROM PERMITS);
```

or, more efficiently,

```sql
SELECT *
FROM PARCELS P
WHERE **NOT EXISTS**

(SELECT NULL
   FROM FIRES F
   WHERE P.PARCELID = F.PARCELID);

**Advanced Queries: Self-Join**

**A table can be joined to itself**

**Example: Find the paper numbers in the URISA database for papers that use both keyword code 601 AND 602.**

The following query does *not* work, because it is not possible for value for a single column in a single row to contain two values at the same time:

```
SELECT PAPER
FROM MATCH
WHERE CODE = 601
AND CODE = 602;
```

This type of query requires a self-join, which acts as if we had two copies of the MATCH table and are joining them to each other.

```
SELECT M1.PAPER
FROM MATCH M1, MATCH M2
WHERE M1.PAPER = M2.PAPER
  AND M1.CODE = 601
  AND M2.CODE = 602;
```

If you have trouble imagining the self-join, pretend that we actually created two copies of MATCH, M1 and M2:

```
CREATE TABLE M1 AS
   SELECT * FROM MATCH;
CREATE TABLE M2 AS
   SELECT * FROM MATCH;
```

Then, we could join M1 and M2:

```
SELECT M1.PAPER
FROM M1, M2
WHERE M1.PAPER = M2.PAPER
  AND M1.CODE = 601
  AND M2.CODE = 602;
```

The self-join allows us to perform this sort of operation without actually having to copy the table. We can just act as if we had two copies.
Now, let's add the titles to the paper numbers:

```sql
SELECT M1.PAPER, T.TITLE
FROM MATCH M1, MATCH M2, TITLES T
WHERE M1.PAPER = M2.PAPER
    AND M1.PAPER = T.PAPER
    AND M1.CODE = 601
    AND M2.CODE = 602;
```

Example: Find the time that passed between a fire on a parcel and all fires occurring within 300 days later on the same parcel

```sql
SELECT F1.PARCELID, F1.FDATE FIRE1, F2.FDATE FIRE2,
    F2.FDATE - F1.FDATE INTERVAL
FROM FIRES F1, FIRES F2
WHERE F1.PARCELID = F2.PARCELID
    AND F2.FDATE > F1.FDATE
    AND F2.FDATE <= F1.FDATE + 300;
```

Note that a number of days can be added to a date.

# The Zoning Variance Database

<table>
<thead>
<tr>
<th>Zoning Variances*</th>
<th>Schema of ZONING table (and listing of related lookup tables)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL examples using zoning variances*</td>
<td>Annotated SQL queries of ZONING table</td>
</tr>
<tr>
<td>1980 Census data (by Boston NSA)*</td>
<td>Schema of 1980 Boston Census data (and related lookup tables)</td>
</tr>
<tr>
<td>Schema of Decision, Use, NSA, Neighbrhd Lookup Tables*</td>
<td>Schema of Lookup tables (second half of Census data web page)</td>
</tr>
<tr>
<td>Sub-Neighborhood lookup table*</td>
<td>The NSA and NEIGHBRHD tables (bottom of Zoning Variance web page)</td>
</tr>
<tr>
<td>Grouping zoning applicants via 'lookup' tables</td>
<td>Annotated SQL queries illustrating</td>
</tr>
</tbody>
</table>

* Kindly refer to Lecture Notes section
use of lookup tables to categorize ownership of properties seeking zoning variances. (These topics are the focus of next week’s lecture and lab #3.)

| Zoning Variance Database Evolution Chart | Stages of evolution of the ZONING variance database |

* Kindly refer to Lecture Notes section