SD202: Databases

SQL language: advanced topics

Antoine Amarilli

Télécom Paris

SQL injection

Common pattern in server applications: build an SQL query from user data:

```
"SELECT name, phone FROM Staff WHERE name='%s'" % name
```

If the user supplies `john` as a value for `name`, we want to run:

```
SELECT name, phone FROM Staff WHERE name='john'
```

But if we get: `john’ UNION ALL SELECT name, password FROM Staff --`

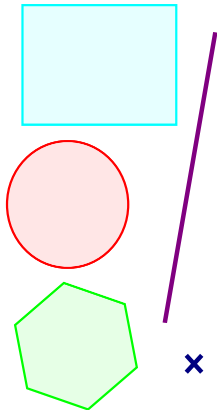
The database will execute:

```
SELECT name, phone FROM Staff WHERE name='john'
UNION ALL SELECT name, password FROM Staff -- '
```

To avoid this, use preparated queries in the host language

```
cursor.execute("SELECT name, phone FROM Staff WHERE name = %s", [('name', name)])
```

Geometric objects



- PostgreSQL has support for geometric objects: points, circles, polygons…
- Efficient support for, e.g., nearest neighbor searches
- Efficient indexes
- PostGIS: PostgreSQL extension to support spatial and geographic objects

Natural language support

- PostgreSQL has some support to index and search **natural language text**:

  - Split text into **tokens**
  - Remove **stop words**
  - Normalize tokens into **lexemes** (stemming)
  - Keeping an **index** of lexemes with their **position** in the text
- All of this is specific to the **language** in use
- There are **more advanced** indexing tools for this job (e.g., Apache Lucene, Apache Solr, ElasticSearch...)

Performance

- Recall that SQL is **declarative**: you specify what you want, not how to obtain it
- A **plan** is a concrete choice of how to implement a query, using tables, indexes, and operators (e.g., intersection)
- The same query can have **different plans** giving the same result but different performance
  - For instance, perform a selection **before** a join if possible (reduces the number of tuples)
  - For instance, joining **multiple tables**: in which order should they be joined?
- You can use `EXPLAIN` to **see** the actual plan in use for a query
- You can use `EXPLAIN ANALYZE` to **time** the actual query execution

Advanced optimizations

You can help PostgreSQL compute the **right execution plans** by:

- Instructing it to create **statistics** on specific tuple subsets
- Updating statistics on tables **manually**
- Writing **joins** in a way that restricts the possible plans

PostgreSQL supports other optimizations:

- **Parallel queries:** evaluating a query using multiple threads
- **Just-in-time compilation:** accelerate the evaluation of `WHERE` clauses

Indexes

- We have seen that declaring a `PRIMARY KEY` or `UNIQUE` constraint would create an **index**
- PostgreSQL makes it possible to **manually declare** additional indexes
- **Tradeoff:** indexes can be useful to speed up some queries, but take space and cost some overhead to maintain
- Several index types:
  - **B-trees:** see later
  - **Hash indexes:** a hash table
  - Indexes for **geometric structures**
  - **Inverted indexes** for composite values (arrays, set of natural language tokens), **block range** indexes…
- The query planner is in charge of **finding the best way** to use indexes