# TinyQueries

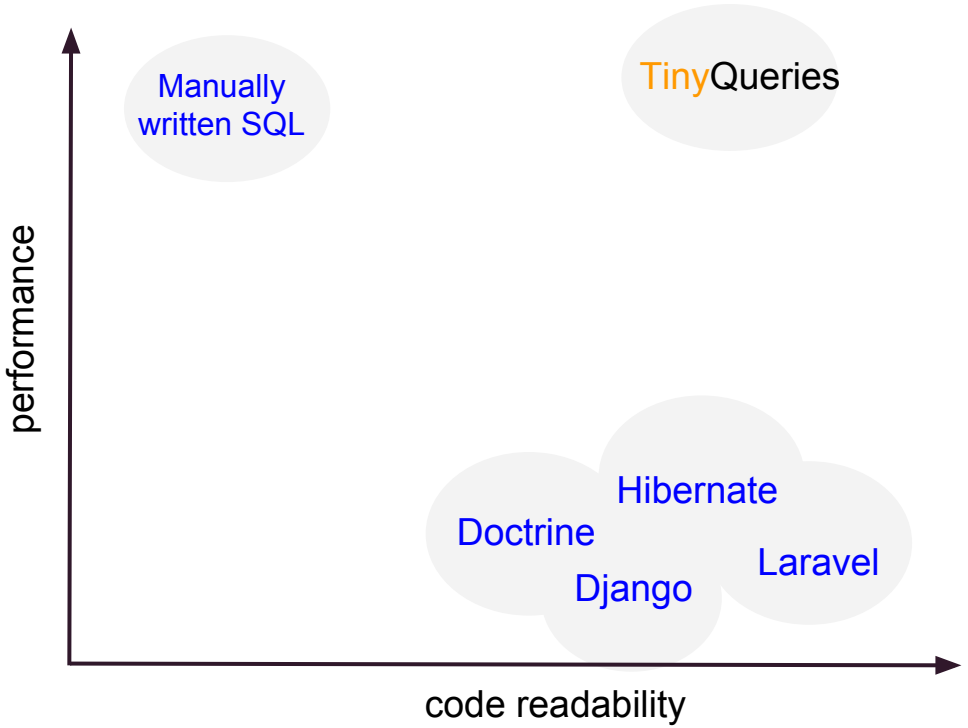
The ultimate query tool for developers

# Why

SQL databases have a huge potential of processing power but current database frameworks don’t use it resulting in unnecessarily slow applications

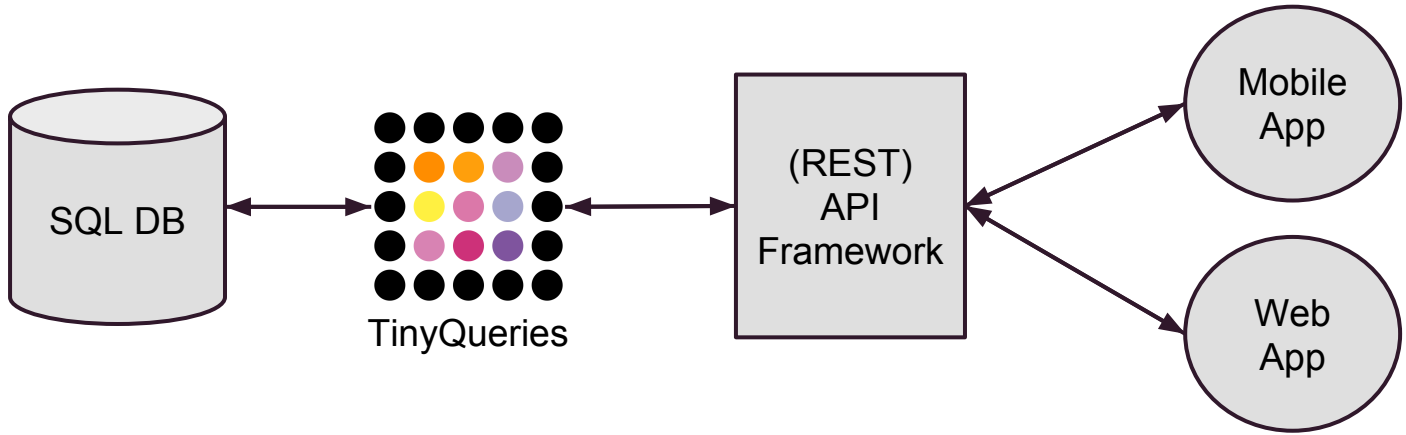
# Best of both worlds

Manually written SQL is fast but hard to read (and write). ORM's like Hibernate make writing queries easy but have bad performance. TinyQueries combines best of both worlds.



# Setup

A typical setup consists of a database, an API framework and TinyQueries in between. TinyQueries translates the endpoints of the API to SQL queries.



# Proven fast!

The *Vrije Universiteit* (Amsterdam) is currently doing research comparing different database frameworks for energy consumption. Preliminary results show that TinyQueries is significantly faster than traditional ORM frameworks, like Propel. The performance of TinyQueries is comparable manually written SQL.

More info regarding the research can be found here

## Preliminary results

### Execution time per table and framework

*95% C.I. testing: SQL does not differ significantly from TinyQueries*
*SQL differs significantly from Propel*

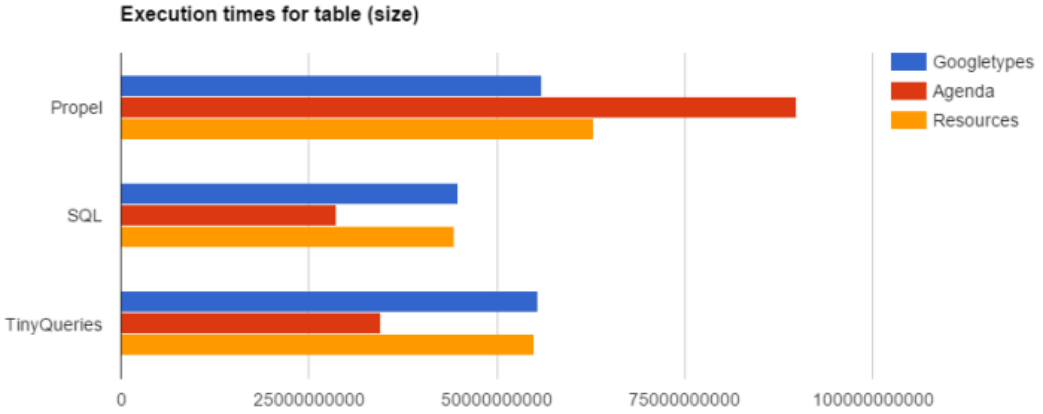
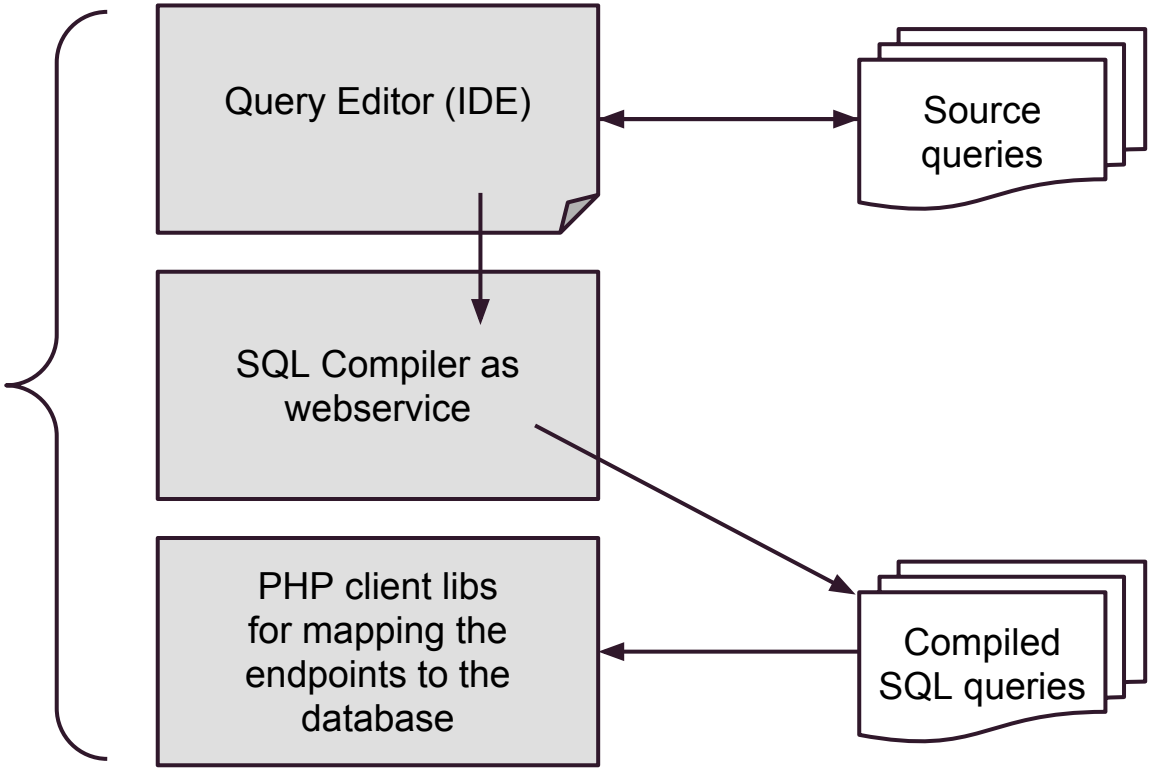



Figure 1.2: Execution time (ns) for every framework when tested with 3 different tables (small = googletypes, medium = agenda, big = resources)

# Components

TinyQueries consists of three components. The first two are development tools - you only need them during development time. The third component is installed on the server and executes the compiled queries.

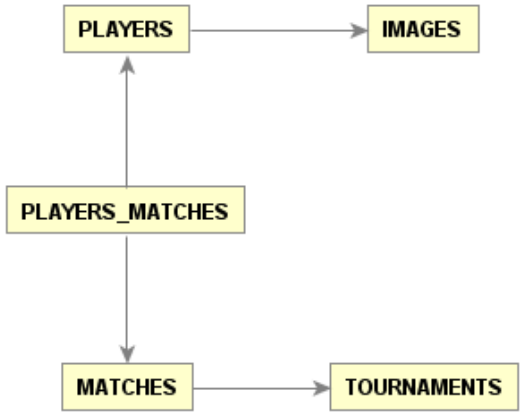


# Example Database

Suppose you have a database which contains data about tennis players and matches. The database scheme is shown to the right.

Suppose you want to get a list of matches and for each match you want to have the tournament and the tennis players and the profile picture of the players.

On the next page the query is shown which you need to write for this.



# Example Query

```
/**
 * Selects matches
 *
 */
{
    select: [
        "this.id",
        "this.result",
        "this.winner.*",
        "this.winner.profilepic.*",
        "this.loser.*",
        "this.loser.profilepic.*",
        "this.tournament.*"
    ],
    from: "MATCHES"
}
```

This query is compiled to

```
select
  M.id as "id",
  M.result as "result",
  P.id as "winner.id",
  P.first_name as "winner.first_name",
  P.last_name as "winner.last_name",
  I.id as "winner.profilepic.id",
  I.url as "winner.profilepic.url",
  I.width as "winner.profilepic.width",
  I.height as "winner.profilepic.height",
  P2.id as "loser.id",
  P2.first_name as "loser.first_name",
  P2.last_name as "loser.last_name",
  I2.id as "loser.profilepic.id",
  I2.url as "loser.profilepic.url",
  I2.width as "loser.profilepic.width",
  I2.height as "loser.profilepic.height",
  T.id as "tournament.id",
  T.name as "tournament.name"
from
  (
    (
      "MATCHES" as M
      inner join
      "TOURNAMENTS" as T
      on M.tournament_fk = T.id
    )
    inner join
    (
      "PLAYERS_MATCHES" as PM2
      inner join
      (
        "PLAYERS" as P2
        left join
        "IMAGES" as I2
        on P2.profilepic_fk = I2.id
      )
      on PM2.player_fk = P2.id
    )
    on (PM2.winner = 0) and (M.id = PM2.match_fk)
  )
  inner join
  (
    "PLAYERS_MATCHES" as PM
    inner join
    (
      "PLAYERS" as P
      left join
      "IMAGES" as I
      on P.profilepic_fk = I.id
    )
    on PM.player_fk = P.id
  )
  on (PM.winner = 1) and (M.id = PM.match_fk)

where
  M.id in (:matchID)
```

The developer defines a query in a JSON-like structure as shown to the left. This query is compiled to SQL as shown to the right.

This example clearly shows the simplicity by which you can generate complex SQL queries

# Some clients


Online platform for creating and selling travel guides. The backend is built on top of TinyQueries


Dutch mental health organization which provides e-health solutions for their clients. A SOAP webservice was built using TinyQueries


Large online store for the hospitality industry. An XML feed for Google Ads was built using TinyQueries

# TinyQueries

Which problems are solved by TinyQueries?

- Stimulates developers to use the potential of a database
- Simplifies writing complex SQL - “*Less for SQL*”
- Compiler can generate vendor specific SQL code so it can be used for any SQL database (DB2, MySQL, MS SQL, Oracle, ..)
- In many cases no programming is needed - just defining queries is enough to create an API
- Performance issues of traditional ORM systems are solved
- Improves code quality - Clean separation of queries and “normal code”