A JIT for SQLite

Carl Friedrich Bolz    Darya Kurilova (CMU)    Laurence Tratt

SOFTWARE DEVELOPMENT TEAM
2015-05-04
Motivation

- SQLite is an embedded database
- The most used database
- Commonly combined with a (dynamic) language
- Getting the data across the boundary is slow
- Can we improve it?

2/15 HTTP://SOFT-Dev.ORG/
Meta-Motivation

- Language composition with actual use cases
- Understand a little bit about DBs
H1 Optimisations which cross the barrier between a programming language and embedded DBMS significantly reduce the execution time of queries.
Hypotheses

**H1** Optimisations which cross the barrier between a programming language and embedded DBMS significantly reduce the execution time of queries.

**H2** Replacing the query execution engine of a DBMS with a JIT reduces execution time of standalone SQL queries.
SQLite

- Small embedded SQL database
- dynamically typed
- used a bit everywhere (Mac OS, Android, ...........)
- reimplementation of Python in RPython
- good JIT via the RPython JIT framework
<table>
<thead>
<tr>
<th>PyPy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python code</td>
</tr>
<tr>
<td>RPy interp.</td>
</tr>
<tr>
<td>Python data</td>
</tr>
</tbody>
</table>

7/15 HTTP://SOFT-Dev.ORG/
def select():
    iterator = conn.execute(""
        "select quantity, extendedprice, discount
        from lineitem"
    )
    sum_qty = 0
    sum_base_price = 0
    sum_disc_price = 0
    for quantity, extendedprice, discount in iterator:
        sum_qty += quantity
        sum_base_price += extendedprice
        sum_disc_price += extendedprice * (1 - discount)
    return sum_qty, sum_base_price, sum_disc_price
PyPy
Python code
RPy interp.
Python data

SQLite
SQL
???
SQL Values
<table>
<thead>
<tr>
<th>PyPy</th>
<th>SQLite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python code</td>
<td>SQL</td>
</tr>
<tr>
<td>RPy interp.</td>
<td>iterate, search ...</td>
</tr>
<tr>
<td>Python data</td>
<td>???</td>
</tr>
<tr>
<td></td>
<td>C interp.</td>
</tr>
<tr>
<td></td>
<td>Algorithms</td>
</tr>
<tr>
<td></td>
<td>SQL Values</td>
</tr>
<tr>
<td></td>
<td>B-Trees</td>
</tr>
</tbody>
</table>

9 / 15 HTTP://SOFT-Dev.ORG/
case OP_Return: {
    pIn1 = &aMem[pOp->p1];
    assert(pIn1->flags == MEM_Int);
    pc = (int)pIn1->u.i;
    pIn1->flags = MEM_Undefined;
    break;
}
case OP_Return: {
    pIn1 = &aMem[pOp->p1];
    assert(pIn1->flags == MEM_Int);
    pc = (int)pIn1->u.i;
    pIn1->flags = MEM_Undefined;
    break;
}

def python_OP_Return(hlquery, op):
    pIn1 = op.mem_of_p(1)
    assert pIn1.get_flags() == CConfig.MEM_Int
    pc = pIn1.get_u_i()
    pIn1.set_flags(CConfig.MEM_Undefined)
    return pc
Optimizations

- language crossing, type conversion
- dynamic typing in SQLite
- inlined user-defined functions and aggregates
Microbenchmarks

- select: 3.02
- innerjoin: 1.52
- pythonjoin: 2.18
- pyfunction: 5.59
- pyaggregate: 4.15
- filltable: 2.73
- geomean: 2.93

SQPyte
SQLite
Summary

- H1 confirmed
- H2 maybe
- Python-SQLite can be improved a lot
Summary

- H1 confirmed
- H2 maybe
- Python-SQLite can be improved a lot

Future Work
- Where exactly is the speedup coming from?
- Interaction with an ORM
- Try with "real" DB?