Two great open source databases: a comparison

Josh Berkus
PostgreSQL Core Team
HP Tux Talks, June 26, 2008
Who is Josh?

• PostgreSQL Core Team
  – 10 years involvement with the project
  – Large database performance, tuning, project press & corporate relations, user groups

• Database geek
  – 15 years database application development
  – MS SQL, MySQL, Oracle, others

• Open Source guru
  – OpenOffice.org, LedgerSMB, Bricolage, OpenBRR, OSCON, more
Topics

- Sound Bite
- History
- Most Common Uses
- Features
- Performance
- Summary
Topics

- Sound Bite
- History
- Most Common Uses
- Features
- Performance
- Summary

mostly about PostgreSQL
Sound Bite

"The most popular open source database"
"The web database"

"The world's most advanced open source database"
"The open source Oracle"
History of MySQL

- MySQL Server development started in 1994, marketed by TCX DataKonsult AB
- MySQL AB founded in 1995 by Michael “Monty” Widenius, David Axmark and Allan Larsson
- Server development based on requirements for practical production use: few features, but fast and stable
- Frequent releases with small changes
- Easy to install and use (15-minute rule)
History of PostgreSQL

- **1986**: POSTGRES at the University of California, Berkeley
  - Michael Stonebraker project
  - Successor to INGRES
- **1994**: first commercialized
  - as Illustra (later merged into Informix)
- **1995**: open-sourced
  - Ported to SQL
  - PostgreSQL Global Development Group formed
- **1997**: ported to Japanese, supported in Japan
- **1999**: first full-time developers & corporate support
- **2004**: native Windows support
- **2006**: supported by Sun
Development History

Designed by/for Application Developers

MySQL

Designed by/for Database Administrators
## Development Priorities

*(historically)*

<table>
<thead>
<tr>
<th>PostgreSQL</th>
<th>MySQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Data integrity</td>
<td>1. Ease-of-use</td>
</tr>
<tr>
<td>2. Security</td>
<td>2. Performance</td>
</tr>
<tr>
<td>3. Reliability</td>
<td>3. Programmer Features</td>
</tr>
<tr>
<td>4. Standards</td>
<td>4. Reliability</td>
</tr>
<tr>
<td>5. DB Features</td>
<td>5. DB Features</td>
</tr>
<tr>
<td>6. Performance</td>
<td>6. Data integrity</td>
</tr>
<tr>
<td>7. Ease-of-use</td>
<td>7. Security</td>
</tr>
<tr>
<td>8. Programmer Features</td>
<td>8. Standards</td>
</tr>
</tbody>
</table>
Development Direction
(a simplification)

MySQL

PostgreSQL

Simple, Easy to Use, Fast

Features, Security, Standards
Community

Owned by one company with user community

Community-owned with many companies involved
• Core MySQL is 100% owned by Sun/MySQL
• 90% of MySQL developers work for Sun
  – except for the many storage engines
• MySQL has a large user community
  – many thousands active worldwide
  – many partners in other open source groups
• Sun/MySQL contributes to other OSS projects
  – PHP especially
• PostgreSQL has a large distributed developer and user community

• Not owned by any one company
  – dozens of companies and individuals contribute code
  – est. over 200 developers in 14 time zones

• "Community Owned"
  supported by 5 different non-profits
Most Common Uses

- Web sites
- CRM
- Logging
- OEM applications
- Telecom (cluster)
- Network tools
- Data Warehouse

- ERP
- Data Warehouse
- Geographic
- Web Sites
- OEM applications
- Network tools
- CRM
Releases

- Feature-based releases
  - new features in minor releases
  - every 1-3 years
    - 3.23: 2000
    - 4.0: 2003
    - 4.1: 2004
    - 5.0: 2005
    - 5.1: 2008?

- Time-based releases
  - no new features in minor releases
  - every year
    - 7.4: 2003
    - 8.0: 2004
    - 8.1: 2005
    - 8.2: 2006
    - 8.3: 2008
Storage Engines

- Pluggable "Storage Engines" allow MySQL to behave like a variety of different databases
  - Telecom DB: MySQL Cluster
  - Non-transactional: MyISAM
  - Transactional: InnoDB
  - Compressed: Archive
  - In-Memory: Memory
  - Write-only: Blackhole
Programmer Features

- Excellent drivers for all languages
  - including JDBC4
- PHP
  - high-performance drivers & special syntax
  - Native driver
- MySQL Proxy
3rd Party Support

• Most open source web projects default to MySQL
  - many use only MySQL
  - primary relational database for most top 25 web sites

• Hundreds of vendors support MySQL
  - more than 50% of multi-database products
  - many "MySQL Partners"
MySQL Scale-Out

- Simple Replication makes it (relatively) simple to scale out
  - used by Google, Yahoo
  - load-balance reads on slaves
  - being supplanted by memcached
Simplicity

- Easy to set up
  - "15 minute rule"
  - everything included
- Easy to administrate
  - programmer-administered
  - most installations don't need tuning
- Easy Replication
  - very simple master-slave & multimaster replication
Features
Migrateability

- Closest to proprietary enterprise DBs
- Automatic migration from Informix
  - Informix is 50% PostgreSQL
- Relatively easy migration from Oracle
  - easiest of any OSS database
  - puts migration cost within affordable range
  - tools for data integration
- SQL Server, DB2 harder
  - but easier that MySQL
"... by default, PostgreSQL is the most security-aware database available ..."

Database Hacker's Handbook
(based on a comparison of PostgreSQL, MySQL, Oracle, DB2 and SQL Server)
Security

- Authentication
  - multiple methods: login, SSL, Kerberos, more
  - host-based authentication

- Logging
  - log output is highly configurable and supports user auditing

- Permissions model
  - SQL ROLES supported, including nested roles
  - multiple settable permissions on all database objects
Security

• Clean code
  – only one security patch per two months
  – community patches usually out in less 72 hours
  – only one exploit in the field in the last four years

• DB Auditing
  – PostgreSQL supports highly configurable triggers and other DB automation
  – No “auditing toolkit” out yet
Transaction Support

• "Bulletproof" ACID thanks to MVCC
  – possibly best of any RDBMS

• Transactional DDL
  – apply schema changes in a transaction
  – great for change management
    • including agile development

• Savepoints
  – spec-compliant "subtransactions"
BI/DW Features

- Large database management features
  - tablespaces, table partitioning
  - automatic large field/row compression
- Powerful query planner & executor
  - complex queries with nested subselects, outer joins and calculated fields
  - large many-table joins with multiple join types
- Data mining features
  - full text indexing and regex support
  - embed external language DM modules
select a12.DAY_OF_WEEK_NBR AS DAY_OF_WEEK_NBR,
    max(TO_CHAR(a12.DATE_DESC ,'Day')) AS CustCol_6,
    a11.DATE_ID AS DATE_ID,
    max(a12.DATE_DESC) AS DATE_DESC,
    a11.FI_ID AS FI_ID,
    max(a13.FI_NAME) AS FI_NAME,
    a12.WEEK_YEAR_ID AS WEEK_YEAR_ID,
    max(a14.SHORT_WEEK_DESC) AS SHORT_WEEK_DESC,
    sum (session_count) AS WJXBFS1,
    sum ( a11_count ) AS WJXBFS2
from ( SELECT DATE_ID, FI_ID, count(distinct SESSION_ID) as session_count, COUNT(*) as a11_count
    FROM edata.WEB_SITE_ACTIVITY_FA
    WHERE DATE_ID in (2291, 2292, 2293, 2294, 2295)
    GROUP BY DATE_ID, FI_ID )
    a11
join edata.DATE_LU a12 on (a11.DATE_ID = a12.DATE_ID)
join edata.DIM_FI a13 on (a11.FI_ID = a13.FI_ID)
join edata.WEEK_LU a14 on (a12.WEEK_YEAR_ID = a14.WEEK_YEAR_ID)
group by a12.DAY_OF_WEEK_NBR,
    a11.DATE_ID,
    a11.FI_ID,
Extensibility

• Create your own database objects
  – almost any db object can be extended easily:
    • functions
    • types
    • operators
    • aggregates
    • pseudo-tables
  – user-created objects are (usually) first class objects
• everything is a function
  – 12 different function languages
CREATE OR REPLACE FUNCTION _choose_random_text (
    thestate _random_text,
    newvalue TEXT )
RETURNS _random_text AS $f$
DECLARE result _random_text;
BEGIN
    result.runcount := COALESCE(thestate.runcount, 0) + 1;
    IF random() < ( 1::FLOAT / result.runcount::FLOAT ) THEN
        result.choice := newvalue;
    ELSE
        result.choice := thestate.choice;
    END IF;
    RETURN result;
END; $f$ LANGUAGE plpgsql;

CREATE AGGREGATE random_agg(
    BASETYPE = text,
    SFUNC = _choose_random_text,
    STYPE = _random_text,
    FINALFUNC = _exit_random_text
);
Special Data

- **Base Types:**
  - char, varchar
  - large text
  - numeric
  - integers
  - floats
  - time, date, timestamp
  - bytea (for binary)

- **Exotic types**
  - geometric: polygon, line
  - GIS (through PostGIS)
  - crypto
  - ISN & ISBN
  - XML
  - network: INET, CIDR
  - arrays
  - full text index
  - genome
Special Data: GIS
**Special Data: genomics**

- BLASTgres, Unison Protein Database

### BLASTgres Results

<table>
<thead>
<tr>
<th>Method Type</th>
<th>Parameter Set</th>
<th># Models in Set</th>
<th>Hits</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMM/Pfam</td>
<td>Pfam_fs 14.0 (set 15)</td>
<td>8</td>
<td>328</td>
</tr>
<tr>
<td></td>
<td>with eval &lt;= 1e-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSSM/PSI-BLAST Profiles (SBP)</td>
<td>PSSM default (set 8)</td>
<td>93</td>
<td>1724</td>
</tr>
<tr>
<td></td>
<td>with eval &lt;= 1e-10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prospect2</td>
<td>Prospect2 ssp_psiPred default (set 23)</td>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>with svm &gt;= 12</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Union (hit by ANY of the above) | 1856 | 17 (89.5%) | 2 (10.5%) | 1839 |
| Intersection (hit by ALL of the above) | 0 | 0 (0.0%) | 0 (0.0%) | 0 |

![3D protein structure diagram]
Procedural Languages

• Use the language you prefer, inside the database:
  – SQL
  – PL/pgSQL
  – C
  – C++
  – Perl
  – Python
  – Java
  – shell
  – R
  – PHP
  – Ruby
  – Tcl

• In beta now: PSM, Lua
create or replace function _set_self_paths ( )
returns trigger as $f$
declare parrec RECORD;
    has_kids BOOLEAN;
begin
    --prevent setting order_by too high
    EXECUTE 'SELECT * FROM ' || TG_RELNAME || ' WHERE id = ' || CAST(NEW.parent as TEXT)
    INTO parrec;
    IF parrec.id is not null THEN
        NEW.path := parrec.path || (NEW.id::TEXT);
        NEW.order_path := parrec.order_path || to_char(NEW.order_by, 'FM0000');
        NEW.show_path := parrec.show_path || ' / ' || NEW.name;
    ELSE
        NEW.path := text2ltree(NEW.id::TEXT);
        NEW.order_path := text2ltree(to_char(NEW.order_by, 'FM0000'));
        NEW.show_path := NEW.name;
    END IF;
RETURN NEW;
end;
$f$ language plpgsql;
CREATE FUNCTION "if_strip_numeric"
(text, smallint) RETURNS text AS $f$
my($the_text, $cutoff) = @_; $the_text =~ s/[\^0-9]//eg;
if ( $cutoff > 0 ) {
    $the_text =
        ( substr $the_text, 0, $cutoff );
} return $the_text;
$f$ LANGUAGE plperl IMMUTABLE, STRICT;
create or replace function statsum(text) returns summarytup as '
  sql<-paste("select id_val from sample_numeric_data ",
                "where ia_id=''", arg1, "''", sep="")
  rs <- pg.spi.exec(sql)
  rng <- range(rs[,1])
  return(data.frame(mean = mean(rs[,1]),
                     stddev = sd(rs[,1]), min = rng[1], max = rng[2],
                     range = rng[2] - rng[1], count = length(rs[,1])))
' language 'plr';
/**
 * Update a modification time when the row is updated.
 */

static void moddatetime(TriggerData td)
throws SQLException
{
    if(td.isFiredForStatement())
        throw new TriggerException(td, "can't process STATEMENT events");

    if(td.isFiredAfter())
        throw new TriggerException(td, "must be fired before event");

    if(!td.isFiredByUpdate())
        throw new TriggerException(td, "can only process UPDATE events");

    ResultSet _new = td.getNew();
    String[] args = td.getArguments();
    if(args.length != 1)
        throw new TriggerException(td, "one argument was expected");

    _new.updateTimestamp(args[0], new Timestamp(System.currentTimeMillis()));
}
HAI
CAN HAS DATABUKKIT?
I HAS A RESULT
I HAS A RECORD
GIMMEH RESULT OUTTA DATABUKKIT "SELECT field
FROM mytable"
IZ RESULT NOOB?
YARLY
BYES "SUMWUNZ IN YR PGSQL STEELIN YR DATA"
KTHX
IM IN YR LOOP
GIMMEH RECORD OUTTA RESULT
VISIBLE RECORD!!FIELD
IZ RESULT NOOB? KTHXBYE
IM OUTTA YR LOOP
KTHXBYE
Hackability

- Clean, easy to read code
- Modular interfaces with clean separation of layers
- #1 most hacked up database
  - Yahoo, Greenplum, Paraccel, Netezza, Truviso ....

```c
/* Forget current agg values */
MemSet(ecxt_aggvalues, 0, sizeof(Datum) * node->numaggs);
MemSet(ecxt_aggnulls, 0, sizeof(bool) * node->numaggs);

/*
 * Release all temp storage. Note that with AGG_HASHED, the hash table is
 * allocated in a sub-context of the aggcontext. We're going to rebuild
 * the hash table from scratch, so we need to use
 * MemoryContextResetAndDeleteChildren() to avoid leaking the old hash
 * table's memory context header.
 */
MemoryContextResetAndDeleteChildren(node->aggcontext);

if (((Agg *) node->ss.ps.plan)->aggstrategy == AGG_HASHED)
{
    /* Rebuild an empty hash table */
    build_hash_table(node);
```
Performance

Better with simple queries and 2-core machines

Better with complex queries and multi-core machines
Benchmarks

- SpecJAppserver 2004, as of July 2007

J2EE Throughput

Acquisition Cost Comparison

Cost in US Dollars

MySQL | PostgreSQL | Oracle
---|---|---

MySQL | PostgreSQL | Oracle
1) Every application performs best with the database for which it was designed.

2) Performance benchmarks for databases are constantly increasing.

3) Top databases are close enough that you can pick the one which suits you best.
Questions?

- e-mail: josh@postgresql.org
- IRC: irc.freenode.net, #postgresql
- blog: blogs.ittoolbox.com/database/soup

Special thanks to:
Giuseppe Maxia and Harrison Fisk of MySQL for information about MySQL