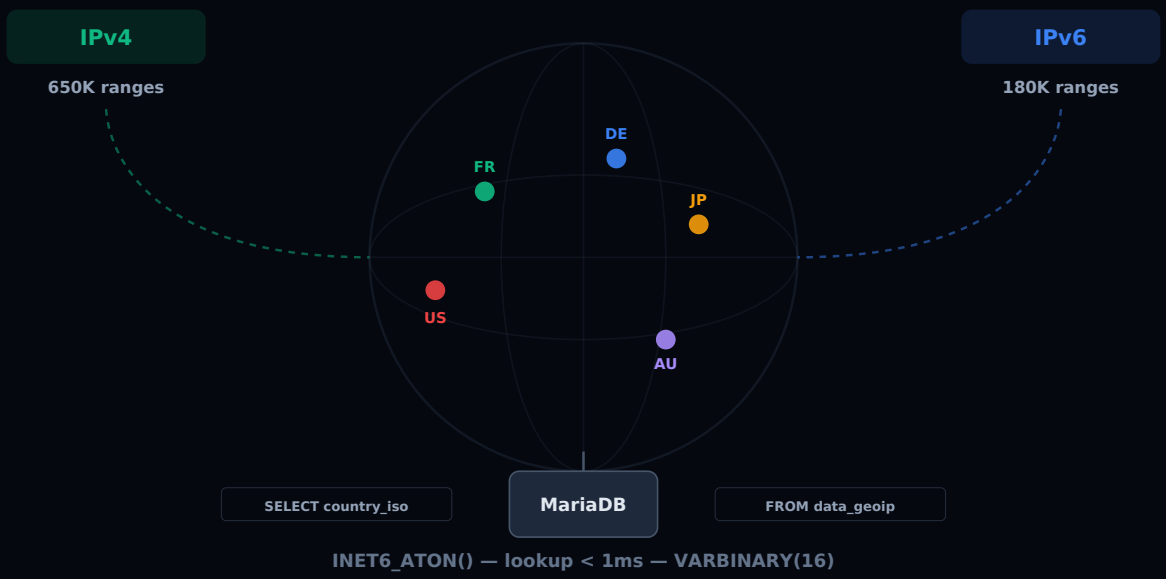
GeoIP in PmaControl: IPv4 and IPv6 Resolution Without Reading the mmdb File on Every Request

Aurélien LEQUOY · April 15, 2026

PMACONTROL GEOIP IPV6 MARIADB PERFORMANCE ARCHITECTURE



The Problem

When you monitor 100+ MariaDB/MySQL servers spread across multiple datacenters and countries, one question comes up every time the main page loads: **where is this server?**

The classic answer: open the MaxMind GeoLite2 `.mmdb` file, perform a lookup for each IP, and display the country flag. Simple... except that:

- Opening a 70 MB `.mmdb` file **on every HTTP request** is expensive
- With 100 servers to resolve, that's 100 file lookups per page
- The file is a binary tree optimized for sequential reads, not for parallel bursts
- In PHP-FPM, each worker reloads the file independently

In PmaControl, the `server/main` page refreshes **every second** via AJAX. Opening the `.mmdb` 100 times per second is nonsensical.

The Solution: Put Everything in MariaDB

The idea is simple: **import all GeoLite2 ranges into a MariaDB table**, then perform standard SQL lookups. A `SELECT` with an index is much faster than traversing a binary tree on disk.

The Schema

```
CREATE TABLE data_geoip (
  id INT AUTO_INCREMENT PRIMARY KEY,
  network_start VARBINARY(16) NOT NULL,
  network_end   VARBINARY(16) NOT NULL,
  country_iso   CHAR(2) NOT NULL DEFAULT '',
  country_name  VARCHAR(100) NOT NULL DEFAULT '',
  INDEX idx_network_start (network_start)
) ENGINE=InnoDB;
```

The choice of `VARBINARY(16)` is crucial:

- **4 bytes** are enough for IPv4 (32 bits)
- **16 bytes** are needed for IPv6 (128 bits)
- `VARBINARY(16)` stores both uniformly
- `INET6_ATON()` converts any IP (v4 or v6) to comparable binary

The Lookup Query

```
SELECT country_iso FROM data_geoip
WHERE network_start <= INET6_ATON('89.30.104.134')
  AND network_end   >= INET6_ATON('89.30.104.134')
LIMIT 1;
```

Result: `FR` (France). Execution time: **< 1 ms**.

The same query works for IPv6:

```
SELECT country_iso FROM data_geoip
WHERE network_start <= INET6_ATON('2001:4860:4860::8888')
```

```
  AND network_end   >= INET6_ATON('2001:4860:4860::8888')
LIMIT 1;
```

Result: `US` (United States — that's Google's public DNS).

The Import: Iterating Over IP Space

IPv4: From 0.0.0.0 to 255.255.255.255

The IPv4 space spans 2^32 = 4.3 billion addresses. We don't iterate one by one — we use `getWithPrefixLen()` from the MaxMind reader, which returns the full CIDR for each address:

```
$ip = 0;
while ($ip <= 4294967295) {
    [$record, $prefixLen] = $reader->getWithPrefixLen(long2ip($ip));

    // Calculate network end
    $networkSize = 1 << (32 - $prefixLen);
    $networkEnd = $ip + $networkSize - 1;

    if ($record && !empty($record['country']['iso_code'])) {
        // INSERT into data_geoip
    }

    // Skip the entire CIDR block
    $ip = $networkEnd + 1;
}
```

The result: **~650,000 ranges** imported in a few seconds. Each range covers an entire CIDR block (e.g., `89.30.104.0/22` → 1,024 addresses in a single row).

IPv6: The 2000::/3 Space

The IPv6 space spans 2^128 addresses — impossible to iterate like IPv4. But routable public addresses live in the `2000::/3` block (Global Unicast), and GeoIP allocations are typically /32 to /48.

The principle is the same: advance by the size of the returned prefix. The difference: we work with 16-byte binary addresses.

```
$current = inet_pton('2000::');
$end6    = inet_pton('3fff:ffff:ffff:ffff:ffff:ffff:ffff:ffff');

while ($current <= $end6) {
    [$record, $prefixLen] = $reader->getWithPrefixLen(inet_ntop($current));
    $endBin = binNetworkEnd($current, $prefixLen);

    if ($record && !empty($record['country']['iso_code'])) {
        // INSERT with UNHEX(bin2hex(...))
    }

    $current = binIncrement($endBin); // +1 in 128-bit arithmetic
}
```

The 128-bit binary arithmetic is implemented in pure PHP (no GMP required):

```
// Increment an IPv6 address by 1
function binIncrement(string $bin): string|false
{
    $bytes = unpack('C16', $bin);
    for ($i = 15; $i >= 0; $i--) {
        $bytes[$i]++;
        if ($bytes[$i] <= 255) return pack('C16', ...$bytes);
        $bytes[$i] = 0; // carry
    }
    return false; // overflow
}
```

Results in Production

Volumes

Table	IPv4	IPv6	Total
`data_geoip` (country)	~650K	~180K	~830K ranges
`data_geoip_city` (city)	~3.7M	~1.2M	~4.9M ranges

Performance

Operation	Time
Country import (IPv4 + IPv6)	~30 seconds
City import (IPv4 + IPv6)	~5 minutes
1 IP lookup	< 1 ms
100 IP lookups (server/main page)	~15 ms total
AJAX refresh (1x/second)	negligible

Display

On PmaControl's main page, each server displays its emoji flag next to the IP:

```
□□□ 89.30.104.134:3306    PIXID-MDB-MASTER1
□□□ 136.243.1.1:3306       Hetzner-Slave
□□□ 210.171.224.1:3306     NTT-Tokyo
□□□ 8.8.8.8:3306           Google-Test
```

Private IPs (10.x, 172.16-31.x, 192.168.x, 127.x) have no GeoLite2 record — the flag is simply absent.

Updates

MaxMind updates GeoLite2 every week. To refresh:

```
# Download the new .mmdb into data/
# Then re-run the import:
php App/Webroot/index.php server loadGeoip       # country (~30s)
php App/Webroot/index.php server loadGeoipCity   # city (~5min)
```

The import does a `TRUNCATE` then reinserts everything. No diff or migration needed — it's a disposable cache.

City Table: Going Further

The `data_geoip_city` table adds region, city, GPS coordinates, and timezone:

```
SELECT country_iso, region_name, city, latitude, longitude, time_zone
FROM data_geoip_city
WHERE network_start <= INET6_ATON('136.243.1.1')
  AND network_end   >= INET6_ATON('136.243.1.1')
LIMIT 1;
```

Result: `DE | Saxony | Falkenstein | 50.4779 | 12.3713 | Europe/Berlin`

This opens the door to server cartography, inter-datacenter latency detection, or simply a richer display in the interface.

Why Not Just a LEFT JOIN?

You might want to do a `LEFT JOIN data_geoip g ON g.network_start <= INET6_ATON(s.ip) AND g.network_end >= INET6_ATON(s.ip)` directly in the server query. The problem: with 650K ranges, this range-join is expensive for the optimizer. We prefer N individual lookups (1 per unique IP) which are instant thanks to the index.

Conclusion

By importing GeoLite2 data into MariaDB, we eliminate the dependency on the `.mmdb` file for every page render. The lookup becomes an indexed `SELECT` at < 1 ms, compatible with both IPv4 and IPv6, and the update process is a simple weekly `TRUNCATE + INSERT`.

The source code is available on GitHub — contributions welcome.