ADVANCED DATABASES

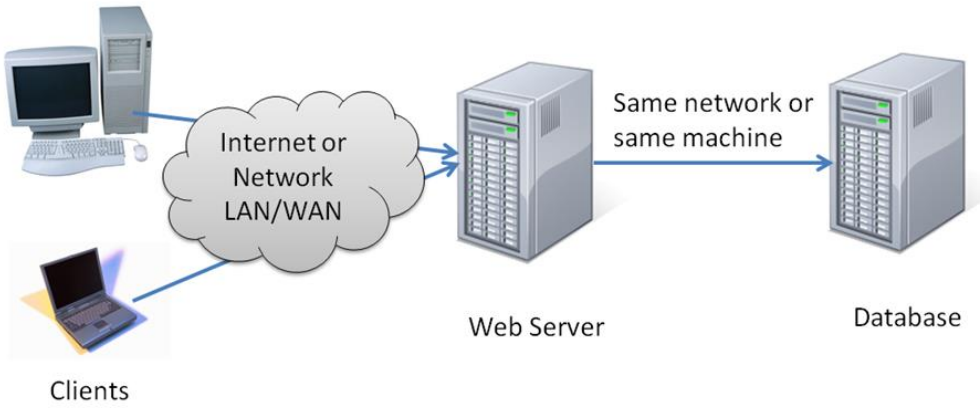
Performance

Wat is performance?

- Response time
  - Queue time
  - Service time
- Throughput
- Aantal gelijktijdige gebruikers

Wat speelt een rol in performance?

- complexiteit van het probleemdomein
  - de data relaties, use-cases, datavolume
- aantal concurrent gebruikers
- hardware-infrastructuur
- netwerkinfrastructuur
- besturingssysteem (instellingen)
- DBMS-instellingen (PostgreSQL server)
- DB sessie-instellingen
- kwaliteit query-optimizer
- transaction isolation levels
- indexen



Hardware

- processor
  - speed vs cores
- RAM geheugen
- opslag (harde schijf etc.)
- netwerk

- voor een hogere performance (kwa hardware): voortdurend bottleneck

DBMS configuratie

- Shared memory
  - is shared postgres mem over meerdere programma's
  - minimaliseert redundante kopieën
  - standaard vaak 32mb (laag), kies ongeveer 10% a 25% van ram om te beginnen
- Effective Cache Size
  - Moet een query result kunnen bewaren
  - Meestal wordt > 50% van ram gekozen
- Max connecties

Vraag

- Hoe zou je een telefoonnummer van een persoon opzoeken in een telefoonboek (bestaande uit 10.000 personen), indien het niet alfabetisch gesorteerd is?

Antwoord

- Brute-force search
- Best-case, hoeveel vergelijkingen?
- Worst-case, hoeveel vergelijkingen?
- Average-case, hoeveel vergelijkingen?
- Nadeel: kost veel energie (tijd)!

In SQL

```
CREATE TABLE telefoonnummers (
      naam              VARCHAR,
      telefoonnummer INTEGER,
);
SELECT telefoonnummer
   FROM telefoonnummers
   WHERE naam = 'Barack Obama';
```

Nadeel: query kost veel tijd

Vraag

- Stel: je moet miljarden keren willekeurige telefoonnummers van personen opzoeken in een ongesorteerde telefoonboek ter grootte van 10.000 personen. Hoe pak je dit aan?

Antwoord

- gesorteerde index op naam
- Binary search (worse case: 13 vergelijkingen)
- Hashing (worse case: ~ 2 vergelijkingen)

Naam	Telefoonnummer
Anton	647643401
Bernard	053468787
Cornelis	987503044
Dirk	138704664
Eduard	301287726
etc...	581465447

In SQL

```
CREATE INDEX tel_naam_index
  ON telefoonnummers (telefoonnummer);
```

Indexes

- Index types (Postgresql):
  - B-tree, Hash, GiST, SP-GiST, GIN
- B-tree: meest gebruikt; ondersteunt > <, patroon en null operators
- Hash: snel; = operator

Multicolumn indexes

```
CREATE TABLE telnums (
      voornaam         VARCHAR,
      achternaamnaam VARCHAR,
      telefoonnummer INTEGER,
);
CREATE INDEX naam_tel_idx
   ON telnums (voornaam, achternaam);
SELECT telefoonnummer
   FROM telnums
   WHERE voornaam = 'Jan'
     AND achternaam ='JANSEN';
```

Postgresql: (momenteel) niet mogelijk voor Hash indexes

Unique index

- Dwingt unieke kolomwaardes af (of combinaties)
- Gebruikt een B-tree

```
CREATE UNIQUE INDEX
   ON telnums (voornaam, achternaam);
```

Nadeel van indexes

- Diskspace
- Kan inserts/updates vertragen
- Moeten regelmatig schoongemaakt worden (Postgresql vacuum)

Query performance tips

- Haalt niet meer rijen op dan nodig

```
SELECT * FROM studs
WHERE jaar = 2
LIMIT 10 OFFSET 20
```

- Minimaliseer de projectie (kolommen in de SELECT clause)
- Rows examined vs rows returned (ratio tussen ~ 1:1 en 10:1)
- Joins vs subqueries. Joins zijn meestal sneller.
- Complexe query vs meerdere kleine queries (voor hetzelfde resultaat) (join decomposition)

Join Decomposition voorbeeld

```
SELECT * FROM tag
  JOIN tag_post ON tag_post.tag_id=tag.id
  JOIN post ON tag_post.post_id=post.id
  WHERE tag.tag = 'informatica';
```

vs

```
SELECT * FROM tag WHERE tag='informatica';
SELECT * FROM tag_post WHERE tag_id=1234;
SELECT * FROM post WHERE post.id
  IN (123,456,567,9098,8904);
```

- Join decomposition voordelen:
  - gemakkelijker te cachen
  - kortere locks
  - queries kunnen efficiënter zijn