Language Technology and Web Applications

Databases II

Johannes Graën

Wednesday 1st November, 2023

Department of Computational Linguistics & Linguistic Research Infrastructure, University of Zurich

What is still missing?

- Θ joins
- data types and domains
- ternary logic
- aggregates and window functions
- indices

Overview

Joins

Data Types & Domains

Aggregates

Indices

Learning Goals for this Week

- You distinguish different types of join operations and their use cases
- You are able to identify the best-matching data type for each attribute
- You know how to analyze data in a database (via aggregations)
- You can explain what indices are good for and know how to use them

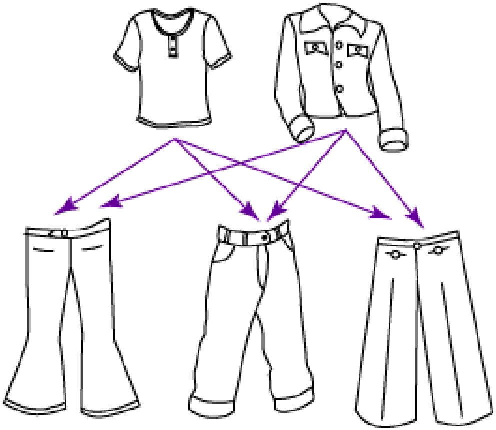
Joins

Joins (recap)

- reconstruct relations between entities by means of foreign and primary keys
- set operations (union, intersection, difference) on partly overlapping tuples
- join conditions are typically using equality (a = b), they are called ‘equi joins’
- in general, Θ join refers to conditions using comparision operations other than ‘=’

Join operations: CROSS JOIN

- also cartesian join
- can be used in conjunction with a ***WHERE*** condition to model every other join type



SELECT statement: Semi Joins and Anti Joins (reprise)

```sql
SELECT * FROM person WHERE EXISTS (
        SELECT 1 FROM driver
        WHERE driver.person_id = person.person_id
);
```

- keywords are *IN*, *EXISTS*, *ANY* / *SOME*, *ALL*
- anti joins are negated semi joins
- for some cases, set operations *UNION*, *INTERSECT* and *EXCEPT* are more efficient than joins

Data Types & Domains

Data Types

- Numeric Types
- Monetary Types
- Character Types
- Date/Time Types
- Boolean Type
- Enumerated Types
- Bit String Types
- UUID Type
- XML Type
- JSON Types
- Arrays
- Domain Types

https://www.postgresql.org/docs/current/datatype.html

Boolean – Ternary Logic

```
SELECT b1, NOT b1 FROM (
    SELECT unnest(ARRAY[TRUE,FALSE,NULL]) b1
) v1;
```

b1	¬ b1
t	f
f	t
¤	¤

Boolean – Ternary Logic

```
SELECT b1, b2, b1 AND b2, b1 OR b2 FROM (
    SELECT unnest(ARRAY[TRUE,FALSE,NULL]) b1
) v1, (
    SELECT unnest(ARRAY[TRUE,FALSE,NULL]) b2
) v2;
```

b1	b2	b1 AND b2	b1 OR b2
t	t	t	t
t	f	f	t
t	¤	¤	t
f	t	f	t
f	f	f	f
f	¤	f	¤
¤	t	¤	t
¤	f	f	¤
¤	¤	¤	¤

Enumerables

```
CREATE TYPE upos AS ENUM
(
    '.',
    'ADJ',
    'ADP',
    'ADV',
    'CONJ',
    'DET',
    'NOUN',
    'NUM',
    'PRON',
    'PRT',
    'VERB',
    'X'
);
```

SERIAL Types

Keyword that triggers

- the creation of a sequence with start value 1 and increment 1
- the creation of an attribute of type *`int`* (*`int2`*, *`int8`*) if type is *`SERIAL`* (*`SERIAL2`*, *`SERIAL8`*)
- the definition of said attribute's default value to *`nextval(<sequence>)`*

Domains

```
CREATE DOMAIN matriculation_number_dom AS char(10)
        CHECK (VALUE ~ '^\d{2}-\d{3}-\d{3}$');
```

http://www.postgresql.org/docs/current/static/sql-createdomain.html

Constraints can also be defined when creating a table:

```
CREATE TABLE student (
    name                  text NOT NULL,
    matriculation_number  matriculation_number_dom CHECK (
    matriculation_number ~ '^13')
);
```

Aggregates

Aggregates

- a typical query returns all results that match
- aggregation functions typically reduce them to a much smaller number
- functions include:
  - count(*)
  - count(<attribute>
  - count(DISTINCT <attribute>)
  - sum()
  - min(), max()
  - avg(), stddev_samp(), stddev_pop()
  - var_samp(), var_pop()
- GROUP BY defines the static part
- HAVING filters aggregated rows using aggregate functions

https://www.postgresql.org/docs/current/functions-aggregate.html

Example (Cantons)

```
CREATE TABLE canton (
    abbreviation   char(2) NOT NULL,  -- official abbreviation
    name           varchar NOT NULL,  -- English canton name
    since          int     NOT NULL,  -- year of joining CH
    population     int     NOT NULL,  -- entire inhabitants
    area           int     NOT NULL,  -- in square kilometres
    alien_ratio    float   NOT NULL   -- ratio of foreigners
);
CREATE TABLE language (
    language_id integer NOT NULL,
    name character varying NOT NULL   -- English language name
);
CREATE TABLE language_in_canton (     -- relation table
    language_id integer NOT NULL,
    canton_id integer NOT NULL
);
```

Example – Query 1

```sql
SELECT abbreviation
FROM canton
JOIN language_in_canton USING (canton_id)
JOIN language USING (language_id)
WHERE language.name <> 'German'
AND since < 1800;
```

- full reference to language.name as ‘name’ is ambiguous
- the attribute ‘since’ is unique in the tuple resulting from the join above

Example (2)

```
SELECT sum(alien_ratio * population) / sum(population)
FROM canton;
```

- *sum( )* is an aggregation function

$$\frac{\sum_i a_i \times p_i}{\sum_i p_i}$$

Example (World)

- *world.dump.sql* contains (outdated) information about countries and their languages
- a dump consists of SQL commands to restore a particular database

We want to answer the following questions:

- How many countries are there?
- On which continents?
- How many head of states are there?
  (And is this number distinct from the number of countries?)
- How many countries are there in Europe?
- How many inhabitant does the country with most inhabitants have?
- How many inhabitant does the country with the smallest/largest population per continent have?
- What os the average population per continent?
- Who reigns more than two countries?
- How many countries on earth speak German?

Example – Query 1

```sql
SELECT count(*)
FROM country;
```

Example – Query 2

```
SELECT DISTINCT continent
FROM country;
```

Example – Query 3

```
SELECT count(DISTINCT headofstate)
FROM country;
```

Example – Query 4

```sql
SELECT count(*)
FROM country
WHERE continent = 'Europe';
```

Example – Query 5

```sql
SELECT max(population)
FROM country;
```

Example – Query 6

```sql
SELECT max(population), min(population)
FROM country
GROUP BY continent;
```

Example – Query 7

```sql
SELECT avg(population), stddev_pop(population)
FROM country
GROUP BY continent;
```

Example – Query 8

```sql
SELECT headofstate, COUNT(*)
FROM country
GROUP BY headofstate
HAVING COUNT(*) > 2;
```

Example – Query 9

```sql
SELECT round(sum(percentage/100*population))
FROM countrylanguage
JOIN country ON countrycode = code
WHERE language = 'German'
```

Example – Query 10

```
SELECT language, round(sum(percentage*population/100)) AS people
FROM countrylanguage
JOIN country ON countrycode = code
GROUP BY language
ORDER BY people DESC
LIMIT 10
```

Indices

Indices

- searching data in a table linearly: $O(n)$
- … feasible for considerably small numbers of record
- searching data in cross-joined tables (self join): $O(n^2)$
- … feasible for considerably small numbers of records of the cartesian product

$\Rightarrow$ linear search is not a good strategy for most queries

Indices

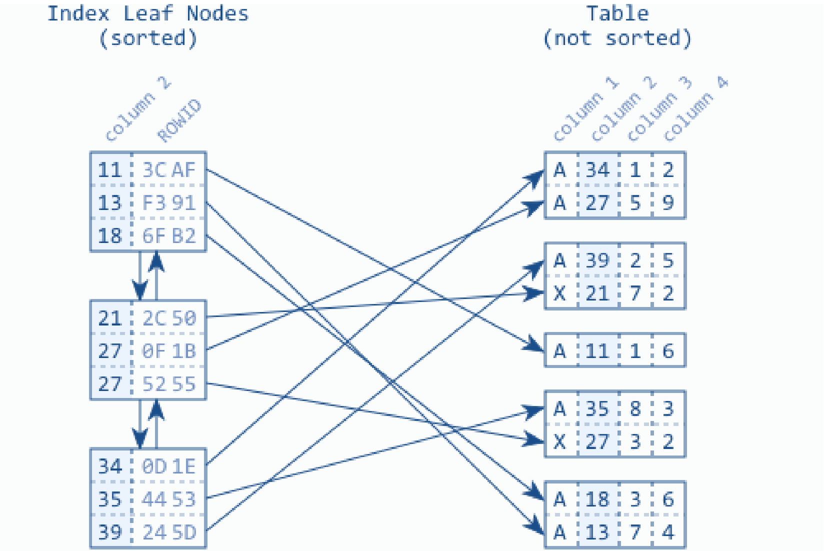
- an index is a data structure that allows for retrieval with sub-linear complexity

Index Types

- B-Tree (balanced tree)
- Hash
- GIN (generalized inverted index)

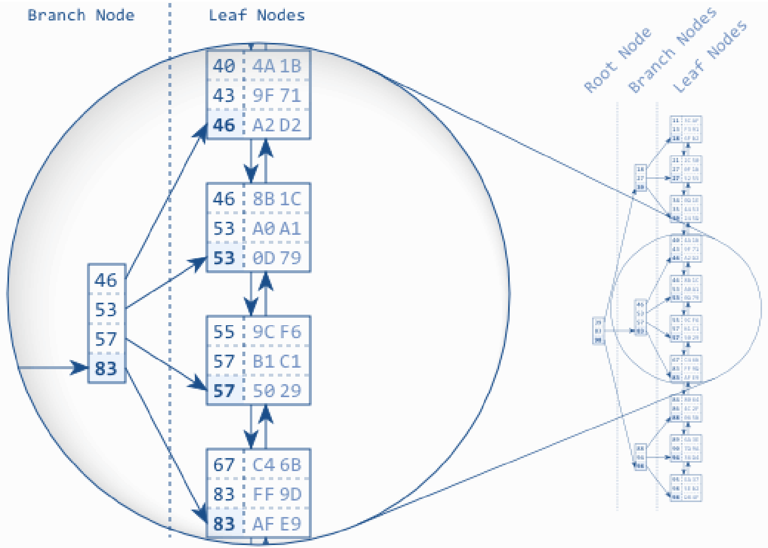
https://www.postgresql.org/docs/current/indexes-types.html

B-Tree (1)



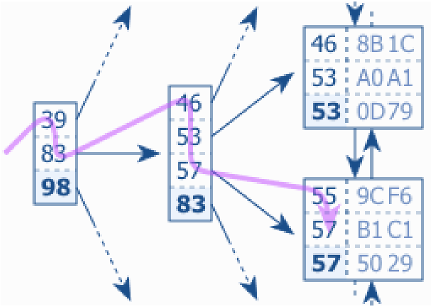
Index Leaf Nodes
(sorted)
Table
(not sorted)
column 2
ROWID
column 1
column 2
column 3
column 4
11 3C AF
13 F3 91
18 6F B2
21 2C 50
27 0F 1B
27 52 55
34 0D 1E
35 44 53
39 24 5D
A 34 1 2
A 27 5 9
A 39 2 5
X 21 7 2
A 11 1 6
A 35 8 3
X 27 3 2
A 18 3 6
A 13 7 4

B-Tree (2)



Branch Node
Leaf Nodes
46
53
57
83
40 4A 1B
43 9F 71
46 A2 D2
46 8B 1C
53 A0 A1
53 0D 79
55 9C F6
57 B1 C1
57 50 29
67 C4 6B
83 FF 9D
83 AF E9
Root Node
Branch Nodes
Leaf Nodes

B-Tree (3)



©Images: *`https://use-the-index-luke.com/`*