Osnove podatkovnih baz

Sistemi za upravljanje podatkovnih baz

Motivacija

- Trajno shranjevanje
- Datoteke?
  - neučinkovite operacije (iskanje, pisanje, ...)
  - omejitev velikosti, nedistribuiranost,
  - oteženo večuporabniško delo, ...
- Podatkovne baze učinkovito naslavljajo vse te probleme ...

DBMS - Database management systems

Zagotavljajo *trajno* shranjevanje *masovnih* podatkovnih zbirk in pri tem poskrbijo za:

- učinkovitost
- zanesljivost
- priročnost
- *varno večuporabniško* shranjevanje in dostopanje do podatkov

Struktura podatkovnih baz

- Podatkovni model (tabele, XML, graf)
- Shema in podatki
- Jezik za opis shem
- Jezik za manipulacijo podatkov

Uporabniki

- Razvijalec DBMS
- Načrtovalec podatkovne baze
- Razvijalec programske opreme
- Skrbniki podatkovne baze (administrator)

Sočasen dostop

- Transakcije: zaporedje operacij, ki se izvedejo v celoti in brez vzporednih operacij s strani drugih uporabnikov
- ACID (Atomicity, Consistency, Isolation, Durability):
  - Atomarnost: transakcije so atomarne
  - Konsistentnost: pretvorba konsistentnega stanja v konsistentno
  - Izoliranost: rezultat transakcije neviden ostalim do zaključka transakcije
  - Stalnost: rezultati transakcij morajo ostati stalni in preživeti "okvare" baz

Vrste sistemov za upravljanje podatkovnih baz

- RDBMS
- Relacijski (MySQL, PostgreSQL, Oracle, SQLite, DB2, MSSQL, ...)
- NoSQL: grafovski (Neo4j, Titan), dokumentni (MongoDB, Elasticsearch), ključ-vrednost (Dynamo, Bigtable), tabelarične (HBase), ...
- Pogovorno rečemo sistemu za upravljanje kar "baza"

Primer - Banka

Banka

- Sestavimo preprosto shemo za banko.
- Kaj rabimo?
  - Osebe: ime, priimek, EMŠO, naslov
  - Račun: številka, lastnik (referenca na osebo)
  - Transakcija: številka, račun (referenca na račun), znesek, opis, čas
- Uporabimo SQLite narečje jezika SQL

Tabela *oseba*

- *ime*, *priimek*: ločimo ime in priimek - bolj kakovostni podatki
- *emso*: vsaka vrstica v tabeli rabi identifikator
  - EMŠO je naravni identifikator, zato ga uporabimo
  - Če se le da, uporabljamo naravne identifikatorje
- *naslov*: poln naslov brez pošte
  - lahko bi ločili ulico, številko (kaj pa poštni predal, čudne številke ...)
- *posta*: uporabimo šifrant pošt in številk - lahko se skličemo na drugo tabelo (glej tabelo *kraj*)

```
CREATE TABLE oseba (
  emso    TEXT PRIMARY KEY,
  ime     TEXT NOT NULL,
  priimek TEXT NOT NULL,
  ulica   TEXT NOT NULL,
  posta   INTEGER NOT NULL REFERENCES kraj(posta)
);
```

Tabela *kraj*

- *posta*: poštna številka; identifikator
- *kraj*: ime kraja

```
CREATE TABLE kraj (
  posta INTEGER PRIMARY KEY,
  kraj  TEXT NOT NULL
);
```

Tabela *racun*

- *stevilka*: številka računa; identifikator; želimo, da se avtomatično generira
- *lastnik*: referenca na lastnika računa (tabela *oseba*, stolpec *emso*)

```
CREATE TABLE racun (
  stevilka INTEGER PRIMARY KEY AUTOINCREMENT,
  lastnik  TEXT NOT NULL REFERENCES oseba(emso)
);
```

Tabela *transakcija*

- *id*: ni naravnega identifikatorja, številčimo sami
- *racun*: sklic na tabelo *racun*, stolpec *stevilka*
- *znesek*: celoštevilski znesek (poenostavitev: "prave" baze bi imele tu računovodski znesek, ki omogoča pravilno zaokroževanje)
- *cas*: čas transakcije; poskrbimo, da se avtomatično generira ob vpisu

```
CREATE TABLE transakcija (
  id      INTEGER PRIMARY KEY AUTOINCREMENT,
  racun   INTEGER NOT NULL REFERENCES racun(stevilka),
  znesek  INTEGER NOT NULL,
  cas     TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  opis    TEXT
);
```

Banka - celoten SQL, narečje SQLite - 1

```
CREATE TABLE kraj (
  posta INTEGER PRIMARY KEY,
  kraj  TEXT NOT NULL
);

CREATE TABLE oseba (
  emso    TEXT PRIMARY KEY,
  ime     TEXT NOT NULL,
  priimek TEXT NOT NULL,
  ulica   TEXT NOT NULL,
  posta   INTEGER NOT NULL REFERENCES kraj(posta)
);
```

Banka - celoten SQL, narečje SQLite - 2

```
CREATE TABLE racun (
  stevilka INTEGER PRIMARY KEY AUTOINCREMENT,
  lastnik  TEXT NOT NULL REFERENCES oseba(emso)
);

CREATE TABLE transakcija (
  id      INTEGER PRIMARY KEY AUTOINCREMENT,
  racun   INTEGER NOT NULL REFERENCES racun(stevilka),
  znesek INTEGER NOT NULL,
  cas     TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  opis    TEXT
);
```

Banka - narečje PostgreSQL - 1

```
--- VERZIJA ZA POSTGRESQL ---

-- tabela krajev s poštnimi številkami
CREATE TABLE kraj (
  posta INTEGER PRIMARY KEY,
  kraj  TEXT
);

-- tabela fizičnih oseb, ki so lastniki računov
CREATE TABLE oseba (
  emso    TEXT PRIMARY KEY,
  ime     TEXT,
  priimek TEXT,
  rojstvo DATE,
  ulica   TEXT,
  posta   INTEGER REFERENCES kraj(posta)
);
```

Banka - narečje PostgreSQL - 2

```
-- tabela računov
-- števec računov, se avtomatično povečuje sam
CREATE SEQUENCE "rstevec" START 100000;

CREATE TABLE racun (
  stevilka INTEGER PRIMARY KEY
           DEFAULT NEXTVAL('rstevec'),
  lastnik  TEXT NOT NULL REFERENCES oseba(emso)
);
```

Banka - narečje PostgreSQL - 3

```
-- tabela vseh transakcij (pologov in dvigov denarja)
CREATE TABLE transakcija (
  id      SERIAL PRIMARY KEY,
          -- tip INTEGER s števcem z začetkom 1
  znesek INTEGER NOT NULL,
  racun  INTEGER NOT NULL REFERENCES racun(stevilka),
  cas    TIMESTAMP NOT NULL DEFAULT NOW(),
  opis   TEXT
);
```