# Projektovanje baza podataka

## ER model

**Entiteti** se predstavljaju pravougaonikom. **Atributi** su u elipsama koje su povezane sa odgovarajućim entitetom. **Kandidat za ključ** se podvlači. **Odnos** se predstavlja rombom i nalazi se između dva entiteta. **Kardinalnost** za posmatrani entitet se piše uz sam entitet u obliku $(l, u)$. **Slab entitet** ne može da postoji samostalno i crta se u duplom pravougaoniku. Kardinalnost kod slabog entiteta je podrazumevano (1, 1). **Generalizacija/specijalizacija** se predstavlja rombom u kome je slovo S. Kardinalnost kod izvedenih entiteta je podrazumevano (1, 1). **Agregacija** je odnos koji istovremeno predstavlja i entitet i predstavlja se kao romb u pravougaoniku. Situacije kada je potrebno agregirati odnos:

- kardinalnost u oba smera je oblika više-više
- odnos ima atribute
- odnos treba da bude u odnosu sa nekim entitetom

## UML dijagram klasa

**Klase** se predstavljaju u vidu tabele, gde se navodi ime klase i lista **atributa**. **Kandidat za ključ** se obeležava stereotipom <<PK>>. **Asocijacija** se predstavlja običnom linijom. **Kardinalnost** za posmatrani entitet se piše uz entitet na drugom kraju asocijacije u obliku $u..l$. **Klasa asocijacije** je asocijacija koja sadrži atribute i predstavlja se kao nova klasa koja je isprekidanom linijom povezana na asocijaciju. $n$**-arni odnosi** se predstavljaju rombom na koji su povezane sve klase koje učestvuju u odnosu. **Generalizacija/specijalizacija** se predstavlja praznim strelicama koje idu od izvedenih klasa ka baznoj. **Agregacija** je tip odnosa u kojem objekat jedne klase predstavlja deo objekta druge klase, pri čemu delovi mogu da postoje nezavisno od celine. Predstavljaju se praznim rombom na strani klase koja je celina. **Kompozicija** je tip odnosa u kojem objekat jedne klase predstavlja deo objekta druge klase, pri čemu delovi ne mogu da postoje nezavisno od celine. Predstavljaju se popunjenim rombom na strani klase koja je celina.

## Prevođenje ER modela i modela klasa u relacioni model

### Prevođenje ER modela

**Redosled prevođenja**:

1. nezavisni entiteti
2. zavisni entiteti i specijalizacije

3. odnosi i agregirani objekti

- **Nezavisni entiteti** se prevode u relacije, a njihovi atributi postaju atributi relacije. Navodimo ime relacije, a u zagradi listu atributa, pri čemu je ključ podvučen.
- **Zavisni entiteti** i **specijalizacije** se prevode u relacije, pri čemu se kao deo ključa dodaje i ključ entiteta od koga zavisi, odnosno bazne klase. Kad god se ključ neke relacije dodaje kao strani ključ novoj relaciji to zapisujemo sintaksom: nova_relacija[strani_ključ] ⊆ postojeća_relacija[primarni_ključ]
- **Odnosi** se prevode sledećim redosledom:
  1. odnosi koji sa bar jedne strane imaju kardinalnost (1, 1), a moguće ih je prevesti - ne pravi se nova relacija, već se entitetu kod koga je (1, 1) dodaje ključ drugog entiteta. Atributi odnosa se dodaju drugom entitetu (gde nije (1, 1)).
  2. odnosi koji sa bar jedne strane imaju kardinalnost (0, 1), a moguće ih je prevesti - pravi se nova relacija u koju se dodaju ključevi oba entiteta, kao i svi atributi odnosa. Primarni ključ će biti ključ entiteta kod koga je (0, 1). Ako je sa obe strane (0, 1), uzima se bilo koji od dva ključa, ali ne oba istovremeno.
  3. odnosi tipa više-više - pravi se nova relacija u koju se dodaju ključevi oba entiteta, kao i svi atributi odnosa. Primarni ključ će biti unija ključeva oba entiteta.

## Prevođenje modela klasa

**Redosled prevođenja**:

1. klase
2. specijalizacije
3. asocijacije i agregirani objekti

- **Klase** se prevode analogno entitetima ER modela.
- **Specijalizacije** se prevode u relacije, pri čemu se kao deo ključa dodaje i ključ bazne klase. Kad god se ključ neke relacije dodaje kao strani ključ novoj relaciji to zapisujemo sintaksom: nova_relacija[strani_ključ] ⊆ postojeća_relacija[primarni_ključ]
- **Odnosi** se prevode sledećim redosledom:
  1. odnosi koji sa bar jedne strane imaju kardinalnost (1, 1), a moguće ih je prevesti - prevode se analogno kao u ER modelu, samo su klase obrnute zbog kardinalnosti.
  2. odnosi koji sa bar jedne strane imaju kardinalnost (0, 1), a moguće ih je prevesti - prevode se analogno kao u ER modelu, samo su klase obrnute zbog kardinalnosti.
  3. odnosi tipa više-više - prevode se analogno kao u ER modelu.
- **Klase asocijacije** se ne prevode u nove relacije, već se atributi dodaju u tabelu već prevedene asocijacije za koju se vezuje.
- **Ternarne veze** se prevode u novu relaciju, gde ključevi klasa koje učestvuju u odnosu postaju atributi relacije. Ako su gornje granice kardinalnosti 1-1-1 kao ključ se uzima bilo koji par ključeva. Ako su gornje granice kardinalnosti 1-1-* kao ključ se uzima ključ klase uz koju je gornja granica * i jedan od ključeva preostale dve klase. Ako su gornje granice

kardinalnosti 1-*-* kao ključ se uzimaju ključevi klasa uz koje je gornja granica *. Ako su gornje granice kardinalnosti *-*-* kao ključ se uzimaju ključevi sve tri klase. Analogno za $n$-arne odnose.
- **Agregacije** se prevode u novu relaciju koja sadrži ključeve klasa koja učestvuju u agregaciji, pri čemu je primarni ključ ključ klase koja predstavlja deo.
- **Kompozicije** se ne prevode u novu relaciju, već se ključ klase koja predstavlja celinu dodaje kao strani ključ klasi koja predstavlja deo.

# Normalizacija

**Kandidati za ključ** se određuju na sledeći način:

- atribut $A$ koji se ne javlja sa desne strane ni jedne FZ je sigurno deo svih kandidata za ključ
- atribut $A$ koji se javlja samo sa desne strane nekih FZ sigurno nije deo ni jednog kandidata za ključ

Nakon toga se određuje zatvorenje potencijalnih kandidata za ključ da bi se odredili stvarni kandidati.

**Normalne forme**:

- **1NF** - svi atributi imaju atomične vrednosti. U našim zadacima ovo uvek važi.
- **2NF** - relacija je u 1NF i nema neključnih atributa koji funkcionalno zavise od pravog podskupa nekog kandidata za ključ.
- **3NF** - relacija je u 2NF i za svaku FZ $X \to A$ važi bar jedno od:
  - $A \in X$
  - $X$ je natključ
  - $A$ je deo nekog ključa
- **BCNF** - relacija je u 3NF i za svaku FZ $X \to A$ važi bar jedno od:
  - $A \in X$
  - $X$ je natključ

**Dekompozicija** relacije $R$ sa atributima $R_a$ po FZ $X \to Y$:

1. $R_1$ sa atributima $X \cup Y$
2. $R_2$ sa atributima $R_a \backslash Y$

# Okidači

**Trigger funkcija** se piše PRE okidača.

```
create or replace function IME_FUNKCIJE
returns trigger as $$
declare
    POMOĆNE_PROMENLJIVE
begin
    TELO_FUNKCIJE
end;
$$ language plpgsql;
```

**Okidač**:

```
create trigger IME_OKIDAČA before|after insert|delete|update on IME_TABELE
for each row
when USLOV
execute function IME_FUNKCIJE (ARGUMENTI)
```

- **before** - telo se izvršava pre operacije. Koristi se za proveru/modifikaciju podataka koji će biti dodati/ažurirani, kao i za proveru konzistentnosti dva reda.
- **after** - telo se izvršava nakon operacije. Koristi se za propagaciju ažuriranja drugim tabelama, proveru konzistentnosti među više tabela, kao i za slanje poruka.

# Okidači nad pogledima

Kreiranje **pogleda**:

```
create view NAZIV_POGLEDA (NAZIVI_KOLONA) as
UPIT;
```

**Okidač**:

```
create trigger IME_OKIDAČA instead of insert|delete|update on IME_TABELE
for each row
when USLOV
execute function IME_FUNKCIJE (ARGUMENTI)
```

# Indeksi

Kreiranje **indeksa**:

```
create index IME_INDEKSA on IME_TABELE (KOLONE) using TIP_INDEKSA;
```

Tipovi indeksa:

- **hash** - kada imamo poređenja po jednakosti ($=$). Najbrži su i zauzimaju najmanje memorije. Ne podržavaju indeksiranje više kolona istovremeno. Najefikasniji su za spajanje tabela. U tom slučaju treba napraviti indeks nad kolonom po kojoj se spajaju i to nad tabelom koja je veća.
- **btree** - kada imamo poređenja po intervalima ($<$, $\leq$, $=$, $\geq$, $>$), kao i operatore "in" i "between". Tabele su male ili srednje veličine. Može se koristiti bilo koji podskup kolona, ali je najefikasniji ako se prvo indeksiraju kolone gde imamo poređenje po jednakosti, pa onda prvo poređenje po nejednakosti. Na primer, ako imamo "where $a = 5$ and $b >= 42$ and $c < 77$" najefikasniji indeks bi bio (a, b).
- **brin** - isto kao btree, ali za velike i sortirane tabele. Efikasnost je uvek ista, redosled nije bitan.

Ako imamo poređenja po više kolona, treba razmotriti da li praviti jedan indeks nad svim kolonama ili više indeksa nad pojedinačnim kolonama. Ako se često javljaju upiti koji se vrše samo nad kolonom $x$ i upiti koji se vrše samo nad kolonom $y$, a samo ponekad upiti koji se vrše nad obe kolone onda je bolje imati odvojene indekse. Možemo da posmatramo i selektivnost uslova. Ako je jedan uslov selektivan, a drugi nije (možda je čak i beskoristan) onda nema potrebe praviti indeks nad obe kolone već samo nad onom čiji je uslov selektivniji. Ako nijedan uslov nije posebno selektivan, možemo napraviti indeks nad obe kolone istovremeno, gde će prva kolona biti ona čiji je uslov selektivniji.

# Planovi izvršavanja

**Selektivnost**:

- $S(A = a) = \frac{1}{card_A(T)}$
- $S(A > a) = \frac{max_A(T) - a}{max_A(T) - min_A(T)}$
- $S(A < a) = \frac{a - min_A(T)}{max_A(T) - min_A(T)}$
- $S(P \wedge Q) = S(P)S(Q)$
- $S(P \vee Q) = S(P) + S(Q) - S(P)S(Q)$

**Kardinalnost**:

- card($T_1$ join $T_2$ on $P$) = $S(P)$ · card($T_1$) · card($T_2$)
- card($T$ filter by $P$) = $S(P)$ · card($T$)

**Broj stranica**:

- **Veličina stranice** je $PS$.

- **Broj redova tabele $T$ koji se mogu smestiti u jednu stranicu** je $PF(T) = \lfloor \frac{PS}{RS} \rfloor$, gde je $RS$ veličina jednog reda koji se dobija sabiranjem veličina pojedinačnih kolona. Za INT se uzima 4B, za VARCHAR($n$) $\frac{n}{2}$B, a za CHAR($n$) $n$B.
- **Broj stranica za zapis tabele** $T$ je $P(T) = \lceil \frac{rows(T)}{PF(T)} \rceil$, gde je $rows(T)$ broj redova tabele $T$.
- Broj stranica nakon **spajanja** zavisi od implementacije:

  1. **block nested loop join**: $P(T_1) + P(T_1) \cdot P(T_2)$, gde je $T_1$ tabela sa manje stranica.
  2. **sort-merge join**: $P(T_1) + P(T_2) + 2 \cdot \lceil P(T_1) \cdot \log_2 P(T_1) \rceil + 2 \cdot \lceil P(T_2) \cdot \log_2 P(T_2) \rceil$
  3. **hash join**: $3 \cdot (P(T_1) + P(T_2))$

  Obratiti pažnju da je nakon spajanja ponovo potrebno izračunati $PF(T)$ jer su sada dve tabele spojene u jednu i povećan je broj potrebnih stranica za skladištenje spojenih tabela.
- Broj stranica nakon **selekcije** jednak je zbiru broja stranica potrebnih da se pročitaju svi redovi tabele i broja stranica potrebnih da se zapišu samo oni redovi koji ispunjavaju uslov selekcije. Odnosno ako imamo tabelu $T$ i prelazimo iz koraka $T_1$ u $T_2$: $\lceil \frac{rows(T1)}{PF(T)} \rceil + \lceil \frac{rows(T2)}{PF(T)} \rceil$. Slično važi i za **projekciju**, samo što se $PF(T)$ menja zbog uklanjanja kolona.
- Broj stranica za **konačni rezultat** jednak je samo broju stranica potrebnih da se pročitaju redovi iz prethodnog koraka, jer se on ne upisuje na disk: $\lceil \frac{rows(T_{prev})}{PF(T)} \rceil$.
- Pored razmatranja različitih implementacija spajanja, još jedna **optimizacija** podrazumeva da prvo izvršimo selekcije i projekcije, a da tek onda vršimo spajanje kako bi samo spajanje, koje je inače najskuplje, manje koštalo.

# Tehnike optimizacije upita

- Izbegavati SELECT * klauzulu.
- Dohvatati samo onoliko redova koliko je potrebno, tj. koristiti LIMIT $n$ OFFSET $m$ na kraju upita.
- Koristiti INNER JOIN umesto Dekartovog proizvoda sa WHERE uslovom.
- Izbegavati LEFT, RIGHT i OUTER JOIN ako je moguće.
- Izbegavati korišćenje funkcija nad kolonama nad kojima se vrši spajanje.
- Izbegavati korišćenje funkcija nad kolonama u WHERE uslovu, već transformisati konstantu sa druge strane uslova. Ako to nije moguće kreirati funkcijski indeks, tj. indeks definisan nad funkcijom kolone.
- Za podupite koristiti EXISTS ako je broj redova rezultata podupita veliki, a IN ako je mali.
- Za proveru da li je tabela prazna koristiti EXISTS umesto COUNT(*) = 0 jer će prvi način prekinuti proveru čim nađe prvo poklapanje.
- Pisati uslove u WHERE umesto u HAVING klauzuli.