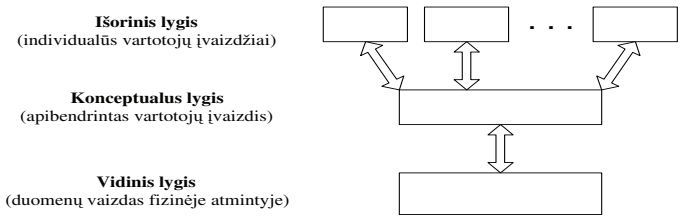
6.5. Duomenų nepriklausomumo lygiai

Sistema palaiko **loginį duomenų nepriklausomumą**, jei vartotojas ir jo programos nepriklauso nuo loginės DB struktūros pasikeitimų.

Fizinis duomenų nepriklausomumas reiškia vartotojų ir jų programų nepriklausomumą nuo fizinės DB struktūros.

ANSI numato **trijų sluoksnių** duomenų bazių sistemų **architektūrą**.



Pav. Trys DBVS architektūros lygiai

- **Vidinis** - susijęs su duomenų saugojimo būdais fizinėse laikmenose.
- **Išorinis** - susijęs su duomenų vaizdavimu individualiems vartotojams. Tai - individualus DB struktūros įvaizdis.
- **Konceptualus** lygis - visų duomenų loginė struktūra. Tai apibendrintas dalykinės srities modelis.

- Loginis duomenų nepriklausomumas – tai nepriklausomumas tarp išorinio ir konceptualaus lygių.
- Fizinis nepriklausomumas – tarp konceptualaus ir vidinio lygių.
- Fizinis nepriklausomumas numato saugomos informacijos pernešimą nuo vienų nešėjų prie kitų, nekeičiant taikomųjų programų.
- Loginis nepriklausomumas leidžia išlikti vartotojų programoms veikiančiomis, netgi pakeitus konceptualų duomenų modelį.
- Fizinis nepriklausomumas pasiekiamas, daugiausia, DBVS ir operacijų sistemos priemonėmis.

6.6. Loginio duomenų nepriklausomumo užtikrinimas

Su loginiu duomenų nepriklausomumu yra susiję du pagrindiniai aspektai – DB struktūros

- **augimas**
- **restruktūrizacija**.

DB **struktūros augimas** galimas dviem būdais:

- papildant egzistuojančią lentelę nauju stulpeliu;
- sukuriant naują lentelę duomenų bazėje.

Naujos lentelės sukūrimas niekaip neįtakoja anksčiau sudarytų užklausų ir kitų SQL sakinių.

Naujas stulpelis lentelėje gali įtakoti tik užklausas

```
SELECT *
```

```
INSERT INTO Seni_Projektai
SELECT * FROM Projektai
```

Naująjį stulpelį galima „**paslėpti**“.

$L(R)$ papildome stulpeliais A:

```
CREATE TABLE L1(R, A)
INSERT INTO L1(R) SELECT R FROM L
DROP TABLE L
CREATE VIEW L(R) AS SELECT R FROM L1
```


DB restruktūrizacija:

$$L(A, B, C) \Rightarrow L_1(A, B) \text{ ir } L_2(A, C)$$

1) sukuriame lenteles $L_1(A, B)$ ir $L_2(A, C)$;
2) iš $L(A, B, C)$ perkeliame duomenis į L_1 ir L_2;
3) sunaikiname lentelę L;
4) sukuriame virtualiąją lentelę:

```
CREATE VIEW L(A, B, C) AS
SELECT L1.A, L1.B, L2.C FROM L1 , L2
WHERE L1.A = L2.A
```


Visiško loginio nepriklausomumo pasiekti nepavyksta.

Loginis duomenų nepriklausomumas turi prasmę tik tuomet, kai DB prieš pakeitimą ir po jo yra tapačios informacijos prasme. Todėl stulpelių ir lentelių naikinimas nevadinamas restruktūrizacija.

6.7. Virtualiųjų lentelių privalumai ir trūkumai

Pagrindiniai virtualiųjų lentelių **privalumai**:

- **Saugumas**. Sukuriant virtualiąsias lenteles galima „paslėpti“ realiųjų lentelių stulpelius bei eilutes.
- **Užklausų paprastumas.** Sudėtingas užklausas ar jų bendrąsias dalis galima apibrėžti virtualiosiomis lentelėmis.
- **Struktūros paprastumas.** Vartojant virtualiąsias lenteles, kiekvienam vartotojui galima sudaryti „savą“ DB struktūrą.
- **Loginis nepriklausomumas**. Restruktūrizuojant DB, galima sukurti vaizdus, kurie „paslepia“ daugelį DB struktūros pasikeitimų.

Virtualiųjų lentelių **trūkumai**:

- **Našumas**. DBVS užklausas virtualiai lentelei turi transformuoti į užklausas realioms lentelėms.
- **Ribojimai atnaujinimui**. Duomenų atnaujinimas yra galimas tik gana paprastose virtualiose lentelėse.

6.8. Materializuotos virtualiosios lentelės

Vykdant užklausas virtualiosioms lentelėms, apibrėžiančioji **užklausa yra vykdoma kiekvieną kartą**.

Kai duomenys konkrečioje VIEW ieškomi dažnai, yra tikslinga **įsiminti** apibrėžiančiosios užklausos rezultatą.

Įsimintas konkretus užklausos rezultatas vadinama **materializuota virtualiąja lentele (MVL)** (**materializuotuoju rodiniu**) (**angl**. *materialized view, materialized query table*).

Paprastai virtualiajai lentelei taikome „lango“ įvaizdį.

MVL galima taikyti „**fotografijos**“ įvaizdį.

Fotografijos **sensta.**

MVL duomenims pasenus, juos reikia atnaujinti.

IBM DB2 ir PostgreSQL MVL sudaromos sakiniu

`CREATE TABLE`.


```
CREATE TABLE ApieVykdytojus(Vykdytojas,
                            VisosValandos, Projektai)
AS  SELECT Vykdytojas, SUM(Valandos),
           COUNT(*)
    FROM Vykdymas
    GROUP BY Vykdytojas
DATA INITIALLY DEFERRED
REFRESH IMMEDIATE
```

`DATA INITIALLY` – pradinio užpildymo strategija

`REFRESH` – MVL duomenų atnaujinimo strategija.

Pradinio užpildymo duomenimis (**`DATA INITIALLY`**) **strategija**:

`DEFERRED` – atidėti vėlesniam laikui (bus atnaujinama sakiniu **`REFRESH TABLE`**).

Duomenų atnaujinimo (**`REFRESH`**) **strategijos**:

- **`REFRESH IMMEDIATE`** – atnaujinti iš karto, kai pasikeičia pradinių lentelių duomenys. **Kiekvienas `INSERT`, `UPDATE` ir `DELETE`** automatiškai iššaukia MVL duomenų atnaujinimą – ne visada tikslinga.
- **`REFRESH DEFERRED`** – duomenys atnaujinami sakiniu **`REFRESH TABLE`**

Duomenų atnaujinimo sakinys

`REFRESH TABLE` <MVL vardas>

Pradiniam duomenų įkėlimui šį sakinį būtina įvykdyti visoms MVL.

Šio sakinio vykdymo dažnis priklauso nuo konkrečių aplinkybių.

Duomenų įvedimas, atnaujinimas ir šalinimas MVL dažniausiai yra neprasmingas ir negalimas, nors kartais tai yra galima ir taikoma.

PostgreSQL: MVL ypatybės

- Sukuriant MVL, ji visada užpildoma duomenimis (**`DATA INITIALLY`** ir **`REFRESH`** neužtikrina).
- Sakinio **`REFRESH TABLE`** nėra.
- Sakiniu **`CREATE TABLE .. AS`** sukurta lentelė tampa paprasta lentele.