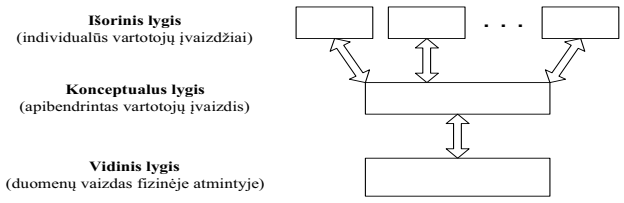
6.5. Duomenų nepriklausomumo lygiai

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** (fizinis) - 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 ar **ryšio tarp jų** (išorinio rakto) **sukūrimas** niekaip neįtakoja anksčiau sudarytų užklausų ir kitų SQL DML 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*).

Įprastai virtualiajai lentelei taikome „lango“ įvaizdį.

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

Fotografijos **sensta.**

MVL duomenims pasenus, juos galima atnaujinti.

PostgreSQL (nuo 9.6 ver.) MVL sudaromos sakiniu:

```
CREATE MATERIALIZED VIEW
```

IBM DB2 MVL sudaromos sakiniu (angl. *materialized query table*)

```
CREATE TABLE AS
```

SQL standarte MVL kūrimo sakinio nėra.


```
CREATE MATERIALIZED VIEW
ApieVykdytojus(Vykdytojas,
               VisosValandos, Projektai)
  AS  SELECT Vykdytojas, SUM(Valandos),
             COUNT(*)
      FROM Vykdymas
      GROUP BY Vykdytojas
  WITH DATA ;
```

čia

WITH DATA – pradinio užpildymo strategija.

[**WITH** [**NO**] **DATA**] – **Pradinio užpildymo duomenimis strategija**:

WITH DATA – užpildoma kuriant MVL (užpildymas gali užtrukti) – numatytoji reikšmė,

WITH NO DATA – atidėti vėlesniam laikui.

Atidėtas užpildymas duomenimis atliekamas sakiniu: **REFRESH MATERIALIZED VIEW**

MVL užpildymas duomenimis vadinamas **materializacija**, kas artima duomenų parankiniam padėjimui (angl. *caching*)

Duomenų atnaujinimo (materializavimo) sakinys:

REFRESH MATERIALIZED VIEW <MVL vardas>

- Šį sakinį būtina įvykdyti visoms MVL, kurios nebuvo užpildytos duomenimis kuriant jas.
- Šio sakinio vykdymo dažnis priklauso nuo konkrečių aplinkybių.
- Duomenų įvedimas, atnaujinimas ir šalinimas MVL dažniausiai yra neprasmingas ir negalimas, nors dalyje DBVS (Oracle) tai yra galima ir taikoma.

Automatinis MVL duomenų atnaujinimas

- Užtikrina dalis DBVS, pvz. DB2 ir Oracle.
- Kuriant MVL, nurodoma duomenų atnaujinimo (**REFRESH**) strategija.
- **REFRESH** strategijos:
  - **IMMEDIATE** – atnaujinti, kai pasikeičia pradinių lentelių duomenys. Sakiniai **INSERT**, **UPDATE** ir **DELETE** automatiškai iššaukia atnaujinimą.
  - **DEFERRED** – duomenys atnaujinami sakiniu **REFRESH**.

PostgreSQL automatinio atnaujinimo **neužtikrina**.

6.9. Laikinosios lentelės

Kuriant lentelę sakiniu **CREATE TABLE** galima nurodyti, kad ji bus laikina:

```
CREATE TEMPORARY TABLE Duomenys (
    Nr      INTEGER NOT NULL PRIMARY,
    Suma    DECIMAL (15, 2)  CHECK(Suma > 0));
INSERT INTO Duomenys
  (Nr, Suma) VALUES (1, 120.20);
INSERT INTO Duomenys
  (Nr, Suma) VALUES (2, 20.10);
```

SELECT * **FROM** *Duomenys*

Nr	*Suma*
1	120.20
2	20.10

SELECT * **FROM** *Duomenys* **WHERE** *Nr* = 1

Nr	*Suma*
1	120.20

Duomenys laikinojoje lentelėje gali išlikti iki transakcijos pabaigos ar iki sesijos pabaigos – tai nurodoma fraze

ON COMMIT DELETE | **PRESERVE ROWS**

```
CREATE TEMPORARY TABLE Duomenys (
    Nr      INTEGER NOT NULL PRIMARY,
    Suma    DECIMAL (15, 2)  CHECK (Suma > 0)
    ON COMMIT DELETE ROWS);
```

– eilutės lentelėje išliks tik iki transakcijos pabaigos.

PRESERVE ROWS – numatytasis veiksmas.

Pasibaigus darbo su DB **sesijai** (atsijungus nuo DB), skirtingos DBVS skirtingai elgiasi su laikinosiomis lentelėmis:

1) PostgreSQL: laikinoji lentelė sunaikinama, norint pasinaudoti kitoje sesijoje reikia perkurti ją.
2) SQL standartas: ištrinami tik duomenys, o lentelės apibrėžimas išlieka, kol lentelė nebus sunaikinta. Kitoje sesijoje ją vėl galima užpildyti duomenimis:

```
INSERT INTO Duomenys
AS SELECT Vykdytojas, SUM(Valandos)
    FROM Vykdymas
```


Laikinoji lentelė gali būti ir sukurta užpildant ją duomenimis:

```
CREATE TEMPORARY TABLE
ApieVykdytojus(Vykdytojas, VisosValandos, Projektai)
AS  SELECT Vykdytojas, SUM(Valandos),
           COUNT(*) FROM Vykdymas ;
```

Šis apibrėžimas tapatus tokiam:

```
CREATE TEMPORARY TABLE
ApieVykdytojus (Vykdytojas,VisosValandos, Projektai)
AS  SELECT Vykdytojas, SUM(Valandos),
           COUNT(*) FROM Vykdymas
ON COMMIT PRESERVE ROWS;
```


Lentelė, sukurta taikant užklausą (**AS SELECT**), neturi tiesioginio ryšio su pradinėmis lentelėmis, todėl neturi duomenų atnaujinimo mechanizmo.