Osnove relacijskih podatkovnih zbirk in jezika SQL

Učni izidi

Po zaključku te enote bodo študenti sposobni:

- Pojasniti temeljne koncepte relacijskih podatkovnih zbirk in njihovo vlogo pri upravljanju strukturiranih podatkov.
- Opisati ključne sestavine sistemov podatkovnih zbirk: tabele, razmerja, ključe in omejitve.
- Razumeti načela normalizacije in jih uporabiti za izboljšanje zasnove podatkovnih zbirk.
- Pisati poizvedbe SQL z uporabo stavkov SELECT, WHERE, GROUP BY, HAVING, JOIN, ORDER BY, INSERT, UPDATE in DELETE.
- Načrtovati osnovne relacijske sheme za nabore okoljskih podatkov (npr. kakovost zraka, spremljanje hrupa).
- Analizirati nabore okoljskih podatkov z jezikom SQL (agregiranje, filtriranje, poizvedbe JOIN in podpoizvedbe).
- Razlagati rezultate poizvedb in pripraviti kratka analitična poročila.

Vsebina predavanja

Uvod

V digitalnih sistemih so podatki temeljno sredstvo. Okoljske agencije, raziskovalne skupine in državne ustanove zbirajo velike količine podatkov opazovanj: meritve ozračja, meteorološke odčitke, kazalnike kakovosti vode, evidence emisij itd. Brez strukturiranega pristopa postanejo takšni podatki razdrobljeni in nedosledni ter jih je težko analizirati.

Relacijske podatkovne zbirke organizirajo informacije v strukturirane tabele, **pri čemer vsaka tabela predstavlja določeno vrsto entitete (npr. postaje**, **meritve**, **uporabnike**).

Podatkovni model temelji na relacijskem modelu, ki ga je uvedel E. F. Codd in ki zagotavlja doslednost ter matematično utemeljenost.

Ključne značilnosti:

1. Struktura, ki temelji na tabelah

- Podatki so shranjeni v tabelah (relacijah) z vrsticami (terkami) in stolpci (atributi).

- Vsaka tabela ima določeno shemo, ki opredeljuje imena atributov, podatkovne tipe in omejitve.

2. Stroga shema

- Vsi podatki morajo biti skladni z vnaprej določeno strukturo.
- Vsak stolpec ima določeno domeno (celo število, besedilo, datum itd.).
- Omejitve, kot so PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE in CHECK, zagotavljajo celovitost podatkov.

3. Strukturirani in dosledni podatki

- Zapisi morajo natančno ustrezati shemi. Primer: tabela *AirQuality(station_id, datetime, pollutant, value)* zahteva, da vsaka vrstica vsebuje te atribute z ustreznimi podatkovnimi tipi.

4. Razmerja med tabelami

- Tabele se lahko medsebojno sklicujejo s ključi, kar omogoča normalizirano zasnovo in preprečuje odvečnost.

Zaradi te strukture so relacijske podatkovne zbirke najprimernejše za dobro opredeljene, strukturirane podatke, pri katerih je doslednost ključna – na primer za finančne sisteme, znanstvene nabore podatkov, upravljanje zalog in spremljanje okolja.

Pregled tehnologije

Temeljni koncepti relacijskih podatkovnih zbirk

Tabele in sheme

V relacijskem modelu so vsi podatki organizirani v logične strukture, imenovane tabele, ki jih formalno imenujemo tudi relacije *(*Tabela 1). Tabela predstavlja eno samo vrsto entitete – na primer merilno postajo, meritev onesnaženosti ali vremensko opazovanje – ter omogoča enoten, strukturiran način shranjevanja več primerkov te entitete. Vsaka tabela je sestavljena iz vrstic in stolpcev. Vrstice oziroma terke zajemajo posamezne zapise, stolpci oziroma atributi pa opisujejo določene lastnosti vsakega zapisa.

Tabela 1. Elementi relacijskega modela in njihova predstavitev

Element relacijskega modela	Oblika predstavitve
Relacija	Tabela
Terka	Vrstica tabele
Atribut	Naslov stolpca tabele
Ključ	Nabor atributov, ki enolično identificira vsako vrstico v tabeli, povezuje tabele ali izboljšuje učinkovitost operacij nad tabelami
Domena	Množica vseh možnih vrednosti, ki jih lahko zavzame atribut
Shema relacije	Seznam imen atributov (naslovov stolpcev tabele), ki določajo strukturo relacije

Oglejmo si na primer tabelo, ki se uporablja v sistemu za spremljanje okolja:

AirQuality(station_id, datetime, pollutant, value, unit).

Vsaka vrstica v tabeli *AirQuality* predstavlja posamezno meritev, opravljeno v določenem trenutku, na določeni postaji in za določeno onesnaževalo. Tabela uveljavlja dosledno strukturo: vse meritve morajo vsebovati identifikator postaje, časovni žig, ime onesnaževala, številčno vrednost in pripadajočo enoto.

Strukturo tabele določa shema. Shema opredeljuje atribute vsake tabele, njihove podatkovne tipe in pravila, ki jih morajo upoštevati. Je načrt podatkovne zbirke in zagotavlja, da so shranjeni podatki smiselni ter razumljivi. V širšem pomenu lahko shema opisuje celotno podatkovno zbirko, saj določa vse tabele ter razmerja in omejitve, ki jih povezujejo. Ta izrecno določena struktura je ena od glavnih prednosti relacijskih podatkovnih zbirk: zagotavlja predvidljivost, zmanjšuje dvoumnost ter omogoča zanesljivo poizvedovanje in analizo.

Shema podatkovne zbirke za spremljanje okolja lahko na primer vsebuje:

- *Stations (station_id, name, region, latitude, longitude)*
- *AirQuality (station_id, datetime, pollutant, value, unit)*
- *Weather (station_id, datetime, temperature, humidity)*

Ključi

Relacijski model se v veliki meri opira na koncept ključev, ki omogočajo enolično identifikacijo zapisov in ohranjanje logičnih povezav med tabelami.

Primarni ključ (PK) je atribut ali kombinacija atributov, ki enolično identificira vsako vrstico v tabeli. Nobeni dve vrstici ne smeta imeti enake vrednosti primarnega ključa, primarni ključi pa ne smejo vsebovati vrednosti NULL. Tabela Stations lahko na primer kot primarni ključ uporablja station_id, s čimer zagotovi, da ima vsaka postaja v naboru podatkov enoličen identifikator. V kompleksnejših tabelah, kot je AirQuality, je mogoče enoličnost doseči s sestavljenim ključem, na primer s kombinacijo (station_id, datetime, pollutant).

Tuji ključ (FK) ustvari logično razmerje med dvema tabelama. To je atribut v eni tabeli, ki se sklicuje na primarni ključ druge tabele. S tem se zagotavlja doslednost podatkov v celotni podatkovni zbirki. Vsak station_id v tabeli AirQuality mora na primer ustrezati obstoječemu station_id v tabeli Stations. Brez tujih ključev bi se lahko v podatkovni zbirki kopičili neveljavni sklici – meritve s postaj, ki ne obstajajo ali so bile izbrisane.

Kandidatni ključi so atributi, ki bi lahko služili kot primarni ključ, ker omogočajo enolično identifikacijo vrstic. Tabela ima lahko več kandidatnih ključev, vendar je za primarni ključ izbran le eden. Sestavljeni ključi pa so ključi, sestavljeni iz več atributov, in se pogosto uporabljajo, kadar noben posamezen atribut sam ne more zagotoviti enoličnosti.

Ključi so bistveni za učinkovitost, zanesljivost in smiselno razlago podatkov. Podpirajo indeksiranje, preprečujejo podvajanje in omogočajo povezovanje med tabelami – zmožnosti, ki so nepogrešljive v obsežnih okoljskih podatkovnih zbirkah.

Razmerja

Razmerja opisujejo, kako so podatki v eni tabeli povezani s podatki v drugi. Relacijski model opredeljuje več vrst binarnih razmerij, od katerih vsako odraža drugačne povezave iz resničnega sveta.

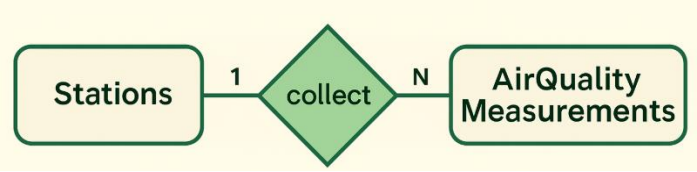
- **Ena proti ena (1 : 1)**
  V razmerju ena proti ena je vsak zapis v eni tabeli povezan z največ enim zapisom v drugi tabeli. Ta vzorec je razmeroma redek, vendar je uporaben, kadar želimo neobvezne, občutljive ali redko uporabljene informacije ločiti v posebno tabelo.

- **Ena proti mnogo (1 : N)**

  Razmerje ena proti mnogo je najpogostejše in najbolj praktično razmerje v relacijskih podatkovnih zbirkah. Pri tej strukturi je lahko ena vrstica v eni tabeli povezana z več vrsticami v drugi.

  Ena postaja za spremljanje okolja lahko na primer vsak dan zbere več sto meritev kakovosti zraka. V tem primeru:

  Tabela Stations je na strani »ena«, tabela AirQualityMeasurments pa na strani »mnogo«.



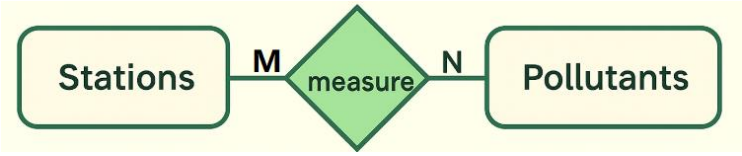
Slika 1. Primer diagrama entiteta–razmerje v Chenovi notaciji za razmerje 1 : N

  Takšna razmerja omogočajo, da se podatkovna zbirka izogne odvečnosti – metapodatki o postaji so shranjeni enkrat, nanje pa se sklicujemo večkrat.

- **Mnogo proti mnogo (M : N)**

  V razmerju mnogo proti mnogo je lahko več vrstic v eni tabeli povezanih z več vrsticami v drugi. Relacijske podatkovne zbirke tega razmerja ne podpirajo neposredno; namesto tega ga uresničimo s povezovalno tabelo, ki shranjuje pare tujih ključev.

  Če lahko postaje na primer spremljajo več onesnaževal, onesnaževala pa se lahko merijo na več postajah, gre za razmerje mnogo proti mnogo (Slika *2*). Povezovalna tabela, kot je *StationPollutant(station_id, pollutant)*, zajema te povezave.



Slika 2. Primer diagrama entiteta–razmerje v Chenovi notaciji za razmerje M : N

Razumevanje razmerij je ključno za načrtovanje normaliziranih shem in pisanje pravilnih poizvedb SQL, zlasti poizvedb JOIN.

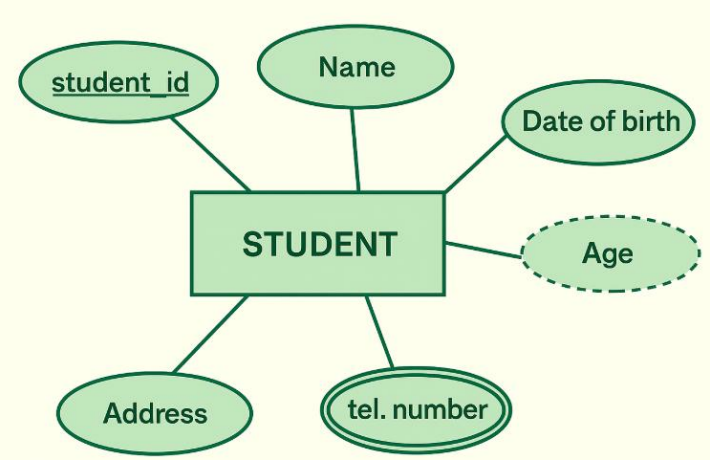
Kardinalnost razmerja izraža število primerkov ene entitete, ki so lahko povezani z enim primerkom druge povezane entitete. V Chenovi notaciji je kardinalnost predstavljena z

ustreznimi števili ob entitetah. Kardinalnost torej kaže, koliko primerkov povezane entitete lahko sodeluje v razmerju.

Atributi

Atribut predstavlja lastnost ali značilnost entitete. Vsak atribut opisuje določen vidik entitete in vsebuje vrednosti, ki pripadajo vnaprej določeni množici. Množica vseh možnih vrednosti, ki jih lahko zavzame atribut, se imenuje njegova domena.

Pri konceptualnem načrtovanju podatkovnih zbirk uporabljamo različne vrste atributov glede na njihovo strukturo, pomen in obnašanje. Glavne vrste atributov so prikazane na Sliki 3 in opisane v Tabeli 2.



Slika 3. Primer diagrama entiteta–razmerje v Chenovi notaciji za entiteto Student

Tabela 2. Vrste atributov

Vrsta atributa	Opis	Primeri
Enostavni (atomarni) atribut	Atribut, ki ga ni mogoče razdeliti na manjše smiselne sestavne dele.	*student_id, priimek*

Sestavljeni atribut	Atribut, ki ga je mogoče razdeliti na manjše podatribute, od katerih ima vsak svoj pomen.	*Naslov → ulica, stavba, stanovanje;* *Polno ime → priimek, ime*
Enovrednostni atribut	Atribut, ki za vsak primerek entitete vsebuje le eno vrednost.	*student_id, številka potnega lista*
Večvrednostni atribut	Atribut, ki lahko za isti primerek entitete vsebuje več vrednosti.	*Telefonske številke (mobilna, domača), e-poštni naslovi*
Izpeljani atribut	Atribut, katerega vrednost se izračuna iz drugih shranjenih podatkov in ni neposredno shranjena v podatkovni zbirki.	*Starost (iz datuma rojstva), število kreditnih točk (iz seznama predmetov)*
Ključni atribut	Atribut ali skupina atributov, ki enolično identificira primerek entitete.	*Primarni ključ: ID študenta; kandidatni ključi: e-poštni naslov, vpisna številka*

Omejitve celovitosti

Omejitve celovitosti so pravila, ki jih podatkovna zbirka samodejno uveljavlja, da ohranja pravilnost, doslednost in zanesljivost. Preprečujejo vnos nepravilnih podatkov v sistem in zagotavljajo veljavnost razmerij med tabelami. Med pogoste vrste omejitev spadajo:

- **NOT NULL**
  Omejitev NOT NULL zagotavlja, da mora stolpec vedno vsebovati vrednost. To je bistveno za atribute, ki so nujni za analizo ali identifikacijo. Časovni žigi okoljskih meritev na primer nikoli ne smejo biti NULL.
- **UNIQUE**
  Omejitev UNIQUE zagotavlja, da nobeni dve vrstici ne vsebujeta enake vrednosti določenega atributa. Uporabna je za identifikatorje, kot so serijske številke, uporabniška imena ali kode onesnaževal.
- **CHECK**
  Omejitve CHECK zagotavljajo, da vrednosti atributov izpolnjujejo logični pogoj.

Omejitev *CHECK* (*value ≥ 0)* na primer zagotavlja, da koncentracij onesnaževal ni mogoče zabeležiti kot negativne, saj bi bilo to znanstveno neveljavno.

- **Referenčna celovitost (s tujimi ključi)**
  Ta omejitev zagotavlja, da se vsaka vrednost tujega ključa ujema z obstoječo vrednostjo primarnega ključa v referenčni tabeli. Preprečuje tudi brisanje vrstic, na katere se sklicujejo drugi zapisi, dokler odvisni zapisi niso ustrezno obravnavani. Referenčna celovitost je bistvena za ohranjanje skladnosti kompleksnih naborov podatkov, zlasti kadar so tabele medsebojno povezane s ključi in razmerji.

Faze načrtovanja podatkovne zbirke

Načrtovanje podatkovne zbirke je strukturiran večstopenjski proces, ki informacijske potrebe iz stvarnega okolja pretvori v učinkovit in zanesljiv sistem za upravljanje podatkov. Čeprav se posamezne metodologije lahko razlikujejo, klasični pristop vključuje štiri temeljne faze:

1. Sistemska analiza področja uporabe
2. Konceptualno načrtovanje
3. Logično načrtovanje
4. Fizično načrtovanje

Vsaka faza obravnava drugo raven abstrakcije in prispeva k izdelavi robustnega, razširljivega ter dobro strukturiranega sistema podatkovnih zbirk.

Proces načrtovanja podatkovne zbirke se začne s sistemsko analizo področja uporabe, ki je temelj vseh nadaljnjih faz načrtovanja. Med sistemsko analizo načrtovalec temeljito spozna resnično okolje, v katerem bo prihodnja podatkovna zbirka delovala. Za to mora podrobno preučiti poslovne procese, znanstveno področje ali organizacijski kontekst, da ugotovi, katere informacije mora sistem shranjevati in kako se bodo uporabljale. Sistemska analiza vključuje tudi prepoznavanje deležnikov – posameznikov, skupin ali oddelkov, ki bodo uporabljali podatkovno zbirko – ter razjasnitev njihovih ciljev, potreb po podatkih in pričakovanj. Za pridobitev teh spoznanj načrtovalci pregledajo obstoječe dokumente, poročila, obrazce, delovne postopke in tehnološka orodja. Analizirajo, kako se podatki trenutno ustvarjajo, obdelujejo, prenašajo in uporabljajo. Rezultat te faze je celovit in strukturiran opis problemskega področja, vključno s ključnimi objekti v njem, njihovimi lastnostmi, medsebojnimi razmerji in pravili, ki določajo njihovo vedenje. To razumevanje je konceptualni temelj celotnega procesa načrtovanja podatkovne zbirke.

Ko je področje temeljito analizirano, proces preide v konceptualno načrtovanje, ki se osredotoča na predstavitev podatkovne strukture na visoki ravni abstrakcije, neodvisno od katere koli tehnologije podatkovnih zbirk. Cilj konceptualnega načrtovanja je izdelati konceptualni podatkovni model – najpogosteje izražen z diagrami entiteta–razmerje (ER) ali diagrami razredov UML –, ki natančno odraža semantiko področja. V tej fazi načrtovalec prepozna entitete, ki se pojavljajo na področju, opredeli atribute, ki jih opisujejo, ter določi razmerja, ki te entitete povezujejo, vključno z njihovimi kardinalnostmi in neobveznostjo. Konceptualno načrtovanje vključuje tudi poslovna pravila, odkrita med sistemsko analizo, da prihodnja podatkovna zbirka odraža omejitve in procese iz resničnega sveta. Osrednje vprašanje na tej točki je, katere informacije naj podatkovna zbirka predstavlja, ne da bi se ukvarjali s tem, kako bodo te informacije nazadnje izvedene ali shranjene.

Naslednja faza, logično načrtovanje, pretvori konceptualni model v strukturo, primerno za izvedbo v določeni vrsti podatkovnega sistema, običajno relacijskem sistemu za upravljanje podatkovnih zbirk (DBMS). Logično načrtovanje je odvisno od izbranega podatkovnega modela, vendar ostaja neodvisno od vidikov fizičnega shranjevanja. Abstraktni konceptualni model se pri tem prevede v relacijske strukture: tabele, primarne ključe, kandidatne ključe, tuje ključe in jasno opredeljene atribute z ustreznimi podatkovnimi tipi. Logično načrtovanje vključuje tudi normalizacijo, ki sistematično odpravlja odvečnost ter preprečuje anomalije pri posodabljanju, vstavljanju in brisanju. Logična shema z omejitvami celovitosti zagotavlja, da bodo podatki ostali pravilni in dosledni. Logično načrtovanje tako odgovarja na vprašanje, kako bo konceptualna predstavitev področja uresničena v podatkovnem modelu izbranega DBMS.

Zadnja faza načrtovanja je fizično načrtovanje, ki obravnava dejansko izvedbo logičnega modela v določenem okolju strojne in programske opreme. Fizično načrtovanje se osredotoča na zmogljivost, učinkovitost in strategije upravljanja podatkov. V tej fazi načrtovalec določi, kako bodo podatki shranjeni in kako se bo do njih dostopalo, ter izbere shranjevalne strukture, organizacijo datotek, strategije indeksiranja in načine particioniranja ali gručenja, ki podpirajo pričakovane delovne obremenitve. Fizični vidiki vključujejo tudi nastavitev dodeljevanja pomnilnika, optimizacijo dostopnih poti in poizvedb ter načrtovanje postopkov varnostnega kopiranja, arhiviranja in obnovitve, da se zagotovita zanesljivost in odpornost proti napakam. Vključeni so lahko tudi varnostni ukrepi, kot so nadzor dostopa, šifriranje in mehanizmi revizijske sledi. Fizično načrtovanje tako prilagodi logično shemo operativnim omejitvam in zmogljivostnim značilnostim sistema DBMS ter obravnava praktično vprašanje, kako bo podatkovna zbirka učinkovito shranjena in kako se bo do nje dostopalo v fizičnem sistemu.

Tabela 3. Povzetek faz načrtovanja podatkovne zbirke

Faza	Ključno vprašanje	Rezultat
Sistemska analiza	*Kaj obsega problemsko področje? Kaj potrebujejo uporabniki?*	Zahteve, poslovna pravila, razumevanje področja
Konceptualno načrtovanje	*Atribut, ki ga je mogoče razdeliti na manjše podatribute, od katerih ima vsak svoj pomen.*	Diagram ER ali konceptualna shema
Logično načrtovanje	*Kako te podatke predstaviti v podatkovnem modelu sistema DBMS?*	Relacijska shema: tabele, ključi, omejitve
Fizično načrtovanje	*Kako učinkovito shranjevati podatke in dostopati do njih?*	Indeksi, particije, nastavitev shranjevanja

Normalizacija

Normalizacija je sistematična metoda, ki se pri načrtovanju relacijskih podatkovnih zbirk uporablja za organiziranje podatkov v dobro strukturirane tabele, s čimer se zmanjšuje odvečnost in preprečujejo nedoslednosti. Zagotavlja, da je vsako dejstvo shranjeno enkrat in samo enkrat ter da je vsak atribut v podatkovni zbirki odvisen od ključa, celotnega ključa in ničesar drugega kot ključa. Normalizacija povečuje preglednost, zmanjšuje potrato prostora za shranjevanje in varuje podatkovno zbirko pred anomalijami pri posodabljanju, vstavljanju in brisanju, ki lahko izkrivijo analitične rezultate.

Čeprav je normalizacija teoretični konstrukt, ki izhaja iz teorije množic in relacijske algebre, je njena vrednost še posebej očitna na praktičnih področjih, kot je upravljanje okoljskih podatkov. Nabori okoljskih podatkov – vključno z meritvami kakovosti zraka, meteorološkimi odčitki, kategorijami onesnaževal in metapodatki o postajah – so pogosto obsežni, ponavljajoči se in medsebojno odvisni. Brez normalizacije lahko podvajanje informacij o postajah, opredelitev onesnaževal ali opisov enot hitro povzroči nedoslednosti med nabori podatkov in postopki poročanja.

Za sistematično odpravljanje odvečnosti se normalizacija izvaja postopoma z zaporedjem normalnih oblik, od katerih vsaka nalaga strožja pravila glede organizacije podatkov.

Najpomembnejše in v praktičnem načrtovanju podatkovnih zbirk najpogosteje uporabljene so prva, druga in tretja normalna oblika.

Prva normalna oblika (1NF)

Prva normalna oblika zahteva, da so vsi podatki v tabeli atomarni, kar pomeni, da vsaka celica vsebuje eno samo, nedeljivo vrednost. Ponavljajoče se skupine, ugnezdene tabele ali več vrednosti v enem polju niso dovoljene.

Slabo zasnovan nabor okoljskih podatkov bi lahko na primer v enem stolpcu shranjeval več onesnaževal:

station_id	pollutants_measured
101	NO2, PM2.5, O3

Takšna predstavitev zelo otežuje poizvedovanje, filtriranje in indeksiranje. V 1NF mora biti vsako onesnaževalo ločeno v svojo vrstico ali premaknjeno v povezano tabelo. Po normalizaciji postane struktura jasna, primerna za poizvedovanje in dosledna:

station_id	pollutants_measured
101	NO2
101	PM2.5
101	O3

Z zagotavljanjem atomarnosti podatkov 1NF postavlja temelj za vse višje normalne oblike.

Druga normalna oblika (2NF)

Druga normalna oblika velja samo za tabele s sestavljenimi primarnimi ključi (ključi, ki jih tvori več kot en atribut). Zahteva, da je vsak neključni atribut odvisen od celotnega primarnega ključa in ne le od njegovega dela. Z drugimi besedami, delne odvisnosti niso dovoljene.

Oglejmo si tabelo, ki shranjuje meritve kakovosti zraka in uporablja ključ *(station_id, datetime)*:

station_id	datetime	pollutant	value	station_name
101	2025-01-15 09:00	NO2	40	Središče Zagreba

2NF zahteva, da takšne atribute premaknemo v ločeno tabelo – v tem primeru *Stations*. Po normalizaciji:

AirQualityMeasurements(station_id, datetime, pollutant, value)

Stations(station_id, station_name)

Ta struktura odpravlja odvečnost in zagotavlja, da se vsak atribut nahaja na ustreznem mestu.

Tretja normalna oblika (3NF)

Tretja normalna oblika odpravlja tranzitivne odvisnosti, kar pomeni, da noben neključni atribut ne sme biti odvisen od drugega neključnega atributa. Atributi morajo biti odvisni samo od primarnega ključa in ne od drugih neključnih atributov.

Za ponazoritev tretje normalne oblike z naborom okoljskih podatkov si oglejmo tabelo, ki shranjuje podatke o vremenskih postajah in regijah, ki jim pripadajo:

station_id	station_name	region_name	region_manager
201	Zahodni rečni breg	Osrednje porečje	dr. Alicia Moreno
202	Južni vrh	Gorsko višavje	inž. Robert Davies

Primarni ključ je *station_id*. Atributi *station_name, region_name* in *region_manager* so vsi neključni atributi. Atribut *region_manager* ni neposredno odvisen od *station_id,* temveč od *region_name* – drugega neključnega atributa. Tako nastane tranzitivna odvisnost:

station_id → region_name

region_name → region_manager.

To krši tretjo normalno obliko.

Za odpravo tranzitivne odvisnosti in doseganje tretje normalne oblike razdelimo tabelo na dve povezani tabeli:

station_id (PK)	station_name	region_name
201	Zahodni rečni breg	Osrednje porečje
202	Južni vrh	Gorsko višavje

region_name (PK)	region_manager
Osrednje porečje	dr. Alicia Moreno
Gorsko višavje	inž. Robert Davies

Podatkovna zbirka je zdaj v tretji normalni obliki:

- *station_id enolično identificira vsako postajo*
- *region_name enolično identificira vsako regijo*
- *region_manager je odvisen samo od svojega ključa (region_name)*
- Noben neključni atribut ni odvisen od drugega neključnega atributa

Normalizacija ima ključno vlogo pri ohranjanju celovitosti in zanesljivosti podatkovne zbirke, saj preprečuje logične anomalije, ki pogosto nastanejo, kadar so podatki shranjeni nestrukturirano ali odvečno. Te anomalije se pojavijo, ker se lahko posamezno dejstvo ponavlja v več vrsticah ali ker so različne vrste informacij neustrezno združene v eni tabeli. V okoljskih informacijskih sistemih, kjer so meritve, metapodatki o postajah, regionalne klasifikacije in regulativni atributi nenehno v medsebojnem stiku, lahko takšne nedoslednosti hitro ogrozijo pravilnost analiz in poročil.

Pogosta težava je anomalija posodabljanja, ki nastane, kadar je treba spremeniti ponavljajoče se informacije. Če se ime merilne postaje ali oznaka regije pojavlja v številnih vrsticah, je treba dosledno posodobiti vsak pojav. Že ena sama pozabljena posodobitev lahko ustvari nasprotujoče si zapise, zaradi česar sistem isto postajo poimenuje z dvema različnima imenoma ali eni regiji dodeli več upravljavcev. Te nedoslednosti se nato prenesejo v poročila, nadzorne plošče in nadaljnje naloge modeliranja.

Druga težava je anomalija vstavljanja, ki nastane, kadar določenih informacij ni mogoče vnesti v podatkovno zbirko, ker druge, z njimi nepovezane informacije še niso na voljo. Novega onesnaževala ali nove okoljske regije na primer morda ni mogoče dodati, dokler ne obstaja vsaj

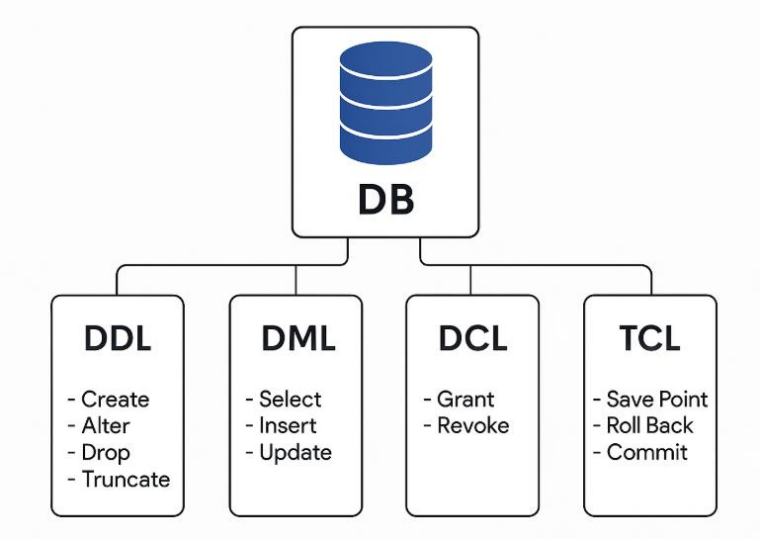
ena pripadajoča meritev. To umetno omejuje vrstni red ustvarjanja podatkov in sistem sili v kršenje naravnih delovnih postopkov.

Anomalija brisanja je nasprotna težava: odstranitev zadnjega pojava nekega operativnega podatka lahko nenamerno odstrani pomembne opisne informacije. Če so metapodatki o postaji ali podrobnosti o regiji shranjeni skupaj z meritvami, lahko izbris zadnje meritve za postajo v celoti odstrani njen opis, čeprav postaja v resničnem svetu še vedno obstaja.

S sistematičnim urejanjem informacij v niz normaliziranih tabel te anomalije odpravimo. Vsako dejstvo je shranjeno natanko enkrat, in sicer v tabeli, kamor logično sodi, vse druge tabele pa se nanj sklicujejo s ključi, namesto da bi ga podvajale. Podatkovna zbirka je zato notranje skladna, lažja za vzdrževanje in odpornejša proti napačnim posodobitvam ali izbrisom. Pri okoljskih uporabah – kot so dolgoročno spremljanje kakovosti zraka, vrednotenje porečij ali ekološko modeliranje – normalizacija zagotavlja, da podatkovni nizi ostanejo zanesljivi, sledljivi in analitično ustrezni. Varuje kontinuiteto znanstvenih opazovanj skozi leta, podpira skladnost s predpisi ter zagotavlja stabilno podlago za obsežno okoljsko modeliranje in odločanje.

Strukturirani jezik za poizvedovanje (SQL)

Strukturirani jezik za poizvedovanje je standardni jezik za delo z relacijskimi sistemi za upravljanje podatkovnih zbirk. Uporabnikom omogoča določanje strukture podatkov, obdelavo shranjenih informacij, urejanje dostopa do objektov podatkovne zbirke in upravljanje transakcij. Čeprav je SQL splošno znan po možnosti pridobivanja podatkov, je v resnici bogat jezik, sestavljen iz več specializiranih podjezikov, od katerih ima vsak poseben namen v življenjskem ciklu operacij nad podatkovno zbirko. Ti podjeziki se tradicionalno razvrščajo v štiri glavne kategorije: DDL, DML, DCL in TCL.



Slika 4: Zgradba SQL

1. Jezik za opredelitev podatkov (DDL)

Jezik za opredelitev podatkov je namenjen opisovanju in spreminjanju strukture objektov podatkovne zbirke. Ukazi DDL določajo shemo – načrt – podatkovne zbirke. Opredeljujejo, katere tabele obstajajo, katere atribute vsebujejo in kako so ti objekti med seboj povezani.

- **CREATE se uporablja za ustvarjanje novega objekta podatkovne zbirke, na primer tabele, pogleda, indeksa ali celo celotne podatkovne zbirke. Ko načrtovalec določi novo tabelo za shranjevanje okoljskih meritev ali doda pogled, ki povzema trende onesnaževal, uporabi ukaz CREATE.**

```
CREATE TABLE Stations (
    station_id INT PRIMARY KEY,
    name VARCHAR(100),
    region VARCHAR(50)
);
```

- **ALTER spremeni obstoječi objekt podatkovne zbirke. To lahko vključuje dodajanje novega stolpca v tabelo, kadar potrebujemo dodatne okoljske atribute, prilagoditev**

podatkovnega tipa za natančnejše meritve ali spreminjanje omejitev zaradi uveljavitve posodobljenih pravil za podatke.

```
ALTER TABLE AirQuality ADD COLUMN unit VARCHAR(20);
```

- **DROP v celoti odstrani objekt podatkovne zbirke. Kadar tabela, indeks ali pogled ni več potreben – morda po prenovi ali združitvi podatkov –, ga ukaz DROP trajno izbriše iz sheme.**

```
DROP TABLE TempData;
```

Ukazi DDL oblikujejo strukturni okvir podatkovne zbirke in določajo temelj, na katerem se podatki shranjujejo in do njih dostopa.

2. Jezik za manipulacijo podatkov (DML)

Medtem ko DDL zgradi strukturo, jezik za manipulacijo podatkov obravnava vsakodnevne operacije nad samimi podatki. Ti ukazi uporabnikom omogočajo vstavljanje novih informacij, pridobivanje obstoječih zapisov, spreminjanje podatkov ter odstranjevanje zastarelih ali napačnih podatkov.

- **INSERT v tabelo doda nove zapise. Ko iz postaj za spremljanje kakovosti zraka prispejo novi odčitki senzorjev ali se vnesejo novi metapodatki za dodatna merilna mesta, ukazi INSERT napolnijo podatkovno zbirko.**

```
INSERT INTO Stations VALUES (101, 'Zagreb-Center', 'Zagreb');
```

- **UPDATE spremeni obstoječe zapise. Če se spremenijo metapodatki postaje – na primer nadmorska višina, popravljene koordinate ali upravljavec –, UPDATE zagotovi, da shranjene informacije ostanejo točne.**

```
UPDATE AirQuality
```

```
SET value = NULL
WHERE value < 0;
```

- **DELETE odstrani zapise iz tabele. Uporablja se, kadar je treba odstraniti napačne vnose ali ko zastareli podatki niso več potrebni.**

```
DELETE FROM Stations WHERE region = 'Unknown';
```

- **SELECT pridobi podatke iz ene ali več tabel, pri čemer pogosto uporabi filtre, razvrščanje, združevanje v skupine in agregiranje. Okoljski analitiki lahko z ukazom SELECT izločijo meritve, ki presegajo zakonsko določene mejne vrednosti, ali primerjajo ravni onesnaževal med merilnimi postajami.**

```
SELECT station_id, pollutant, value
FROM AirQuality
WHERE pollutant = 'PM2.5';
```

  - ***Razvrščanje***

```
ORDER BY value DESC;
```

  - ***Filtriranje z logičnimi operatorji***

```
WHERE value > 50 AND pollutant = 'NO2';
```

  - ***Agregiranje in GROUP BY (uporablja se za okoljsko poročanje)***

```
SELECT station_id, pollutant, AVG(value) AS avg_value
FROM AirQuality
GROUP BY station_id, pollutant;
```

  - ***HAVING – filtriranje agregiranih podatkov***

```
HAVING AVG(value) > 40;
```

  - ***INNER JOIN***

```
SELECT s.name, a.pollutant, a.value
FROM AirQuality a
JOIN Stations s ON a.station_id = s.station_id;
```

  - ***LEFT JOIN***

Primer: prikazati želimo vse merilne postaje, tudi tiste, za katere še ni zabeležena nobena meritev. Tabela 1: *Stations(station_id, station_name).* Tabela 2: *AirQualityMeasurements(station_id,* pollutant, value)

```
SELECT s.station_id, s.station_name, m.pollutant, m.value
FROM Stations AS s
LEFT JOIN AirQualityMeasurements AS m
ON s.station_id = m.station_id;
```

- ***RIGHT JOIN***

  Primer: prikazati želimo vsa merjena onesnaževala, tudi če nekatera še niso dodeljena nobeni postaji. Tabela 1: *AirQualityMeasurements(station_id, pollutant, value).* Tabela 2: *Pollutants(pollutant, description)*

```
SELECT m.station_id, p.pollutant, p.description, m.value
FROM AirQualityMeasurements AS m
RIGHT JOIN Pollutants AS p
ON m.pollutant = p.pollutant;
```

- **Podpoizvedbe**

  Iterativne in ugnezdene analize, npr. preseganje mejnih vrednosti EU:

```
SELECT *
FROM AirQuality
WHERE value > (
      SELECT threshold
      FROM EU_Standards
      WHERE pollutant = AirQuality.pollutant
);
```

DML je operativno jedro jezika SQL, saj uporabnikom omogoča neposredno delo s shranjenimi informacijami.

3. Jezik za nadzor podatkov (DCL)

Jezik za nadzor podatkov ureja dostop do objektov podatkovne zbirke in upravlja dovoljenja. V večuporabniških okoljih podatkovnih zbirk – zlasti v znanstvenih in državnih sistemih, kjer je treba okoljske podatke deliti, hkrati pa tudi zaščititi – DCL zagotavlja, da lahko določena dejanja izvajajo samo pooblaščene osebe.

- **GRANT uporabnikom ali vlogam dodeli dovoljenja za izvajanje določenih operacij nad izbranimi objekti. Raziskovalcem se lahko na primer dodeli dovoljenje za branje podatkov o onesnaževalih, ne pa tudi za spreminjanje metapodatkov postaj.**

```
GRANT SELECT
ON AirQualityMeasurements
TO researcher_1;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE
ON Stations
TO env_admin_role;
```

- **REVOKE odstrani predhodno dodeljena dovoljenja. To je bistveno, kadar se odgovornosti spremenijo ali ko je treba zaradi varnosti oziroma skladnosti s predpisi omejiti dostop do podatkov.**

```
REVOKE INSERT
ON AirQualityMeasurements
FROM researcher_1;
```

- **DENY izrecno prepove določena dejanja in prevlada nad dovoljenji, ki bi jih sicer omogočala. Gre za strog ukrep, ki uveljavlja varnostne politike, tudi če obstaja več ravni dovoljenj.**

```
DENY DELETE
ON Pollutants
TO researcher_2;
```

Ukazi DCL tako pomagajo ohranjati zaupnost in celovitost podatkov ter nadzorovano sodelovanje med skupinami.

4. Jezik za nadzor transakcij (TCL)

Jezik za nadzor transakcij zagotavlja, da se skupine povezanih operacij DML izvajajo atomarno – to pomeni, da bodisi v celoti uspejo bodisi spodletijo, ne da bi za seboj pustile delne spremembe. Transakcije so še posebej pomembne pri obravnavi medsebojno povezanih okoljskih podatkovnih nizov, pri katerih lahko nedoslednosti ogrozijo rezultate analiz.

- **COMMIT zaključi transakcijo in trajno uveljavi vse spremembe, izvedene od zadnje potrditve. Po potrditvi postanejo vstavljene meritve, posodobljeni metapodatki ali izbrisani zapisi del uradnega podatkovnega niza.**

```
BEGIN TRANSACTION;

INSERT INTO AirQualityMeasurements (station_id, pollutant, value)
VALUES (101, 'PM2.5', 23.5);

COMMIT;
```

- **ROLLBACK razveljavi vse spremembe, izvedene med trenutno transakcijo. Če se med posodabljanjem več tabel pojavi napaka – na primer pri povezovanju nove merilne postaje z več referenčnimi tabelami –, ROLLBACK zagotovi vrnitev podatkovne zbirke v stabilno stanje.**

```
BEGIN TRANSACTION;

UPDATE AirQualityMeasurements
SET value = -10     -- neveljavna vrednost
WHERE measurement_id = 455;
```

```
-- Zaznana napaka → razveljavi vse
ROLLBACK;
```

- **SAVEPOINT znotraj transakcije ustvari poimenovane oznake, ki omogočajo delne razveljavitve. To je uporabno pri zapletenih operacijah, kot je uvoz velikih paketov senzorskih podatkov, ko se mora sistem morda vrniti v vmesno stanje, namesto da bi postopek začel znova.**

```
BEGIN TRANSACTION;

UPDATE Stations
SET region = 'Central Basin'
WHERE station_id = 101;

SAVEPOINT sp1;

UPDATE Stations
SET region = 'Mountain Highlands'
WHERE station_id = 102;

-- Razveljavi samo drugo posodobitev
ROLLBACK TO sp1;

COMMIT;
```

S pomočjo TCL SQL zagotavlja skladnost in zanesljivost podatkov tudi pri zapletenih večstopenjskih operacijah.

Lastnosti transakcij ACID

Okoljski podatkovni nizi – meritve kakovosti zraka, meteorološka opazovanja, dnevniki senzorjev in regionalni metapodatki – so pogosto medsebojno povezani in se nenehno posodabljajo. Če je poškodovan ali le delno posodobljen že en del teh podatkov, so lahko ogroženi rezultati analiz in dolgoročni zapisi spremljanja stanja.

Da bi zagotovili predvidljivo in dosledno izvajanje večstopenjskih operacij nad podatkovno zbirko, sodobni sistemi podatkovnih zbirk uporabljajo transakcije – skupine operacij, ki se morajo izvesti kot ena sama, enotna celota. Pravilnost teh transakcij zagotavljajo štiri temeljna načela, skupaj znana kot lastnosti ACID: atomarnost, konsistentnost, izolacija in trajnost.

1. Atomarnost

Atomarnost zahteva, da se transakcija obravnava kot nedeljiva enota: bodisi uspejo vse njene operacije bodisi se ne uveljavi nobena.

Transakcija lahko vključuje veliko korakov, kot so vstavljanje nove meritve, posodabljanje zapisov o umerjanju senzorjev in beleženje dogodka. Atomarnost zagotavlja, da podatkovna zbirka nikoli ne ostane v stanju, v katerem so dokončani le nekateri od teh korakov. Če pride do napake ali sistem odpove sredi postopka, se celotna transakcija razveljavi in podatkovna zbirka se vrne v prejšnje stabilno stanje.

Primer. Postaja za spremljanje kakovosti zraka pošlje paket meritev. Sistem mora vstaviti vse meritve. Če zaradi napake v obliki spodleti že ena sama, atomarnost zagotovi, da se ne vstavi noben zapis, s čimer se preprečijo delni in zavajajoči podatki v podatkovni zbirki.

Atomarnost varuje celovitost zapletenih operacij in preprečuje delne posodobitve, ki bi lahko izkrivile okoljske analize.

2. Konsistentnost

Konsistentnost zagotavlja, da vsaka transakcija spremeni podatkovno zbirko iz enega veljavnega stanja v drugo, pri čemer ohrani vsa določena pravila, omejitve in razmerja.

Ta pravila vključujejo podatkovne tipe, primarne in tuje ključe, omejitve preverjanja ter logiko znanstvenega preverjanja veljavnosti. Konsistentna podatkovna zbirka ta pravila vedno upošteva pred vsako transakcijo in po njej.

Primer. Če podatkovna zbirka uveljavlja pravilo, da koncentracije onesnaževal ne smejo biti negativne, mora transakcija, ki poskuša vstaviti vrednost –15, spodleteti. Podatkovna zbirka zavrne neveljavne podatke, da ohrani znanstveno točnost.

Konsistentnost zagotavlja, da okoljski podatki nikoli ne kršijo fizikalnih, zakonskih ali logičnih omejitev.

3. Izolacija

V resničnih sistemih številni uporabniki in procesi hkrati dostopajo do podatkovne zbirke. Izolacija zagotavlja, da izvajanje ene transakcije ne ovira druge. Dokler transakcija ni dokončana, so njeni vmesni rezultati drugim skriti.

Brez izolacije bi lahko dve transakciji, ki posodabljata iste metapodatke postaje ali vstavljata prekrivajoče se nize meritev, vplivali druga na drugo in povzročili nepredvidljive rezultate.

Primer. Dva raziskovalca hkrati posodabljata različne atribute tabele *Stations*. Izolacija zagotavlja, da si njune posodobitve ne nasprotujejo, se medsebojno ne prepišejo in ne ustvarijo nedoslednih metapodatkov.

Večina sistemov podatkovnih zbirk podpira več ravni izolacije, na primer:

- Read Uncommitted (branje nepotrjenih podatkov)
- Read Committed (branje potrjenih podatkov)
- Repeatable Read (ponovljivo branje)
- Serializable (serializabilno izvajanje)

Višje ravni zagotavljajo večjo zanesljivost, vendar lahko zmanjšajo sočasnost. V okoljskih sistemih, ki sprejemajo velike pakete senzorskih podatkov, je bistveno najti pravo ravnovesje.

4. Trajnost

Trajnost zagotavlja, da po potrditvi transakcije njeni rezultati postanejo trajni – tudi ob okvari strojne opreme, izpadu električne energije ali sesutju sistema.

Sistemi podatkovnih zbirk trajnost zagotavljajo z:

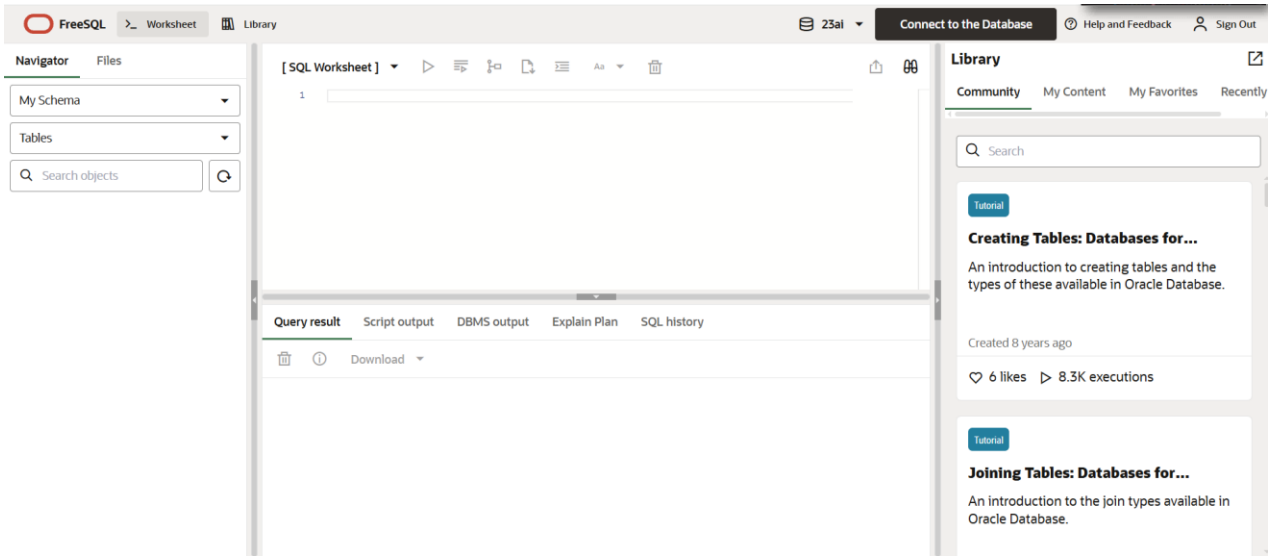
- dnevniki vnaprejšnjega zapisovanja,
- ustvarjanjem kontrolnih točk,
- redundantno hrambo,
- samodejnimi mehanizmi obnovitve.

Primer. Po uspešnem vstavljanju meteoroloških odčitkov za cel dan in izvedbi ukaza COMMIT morajo podatki ostati varno shranjeni, tudi če le nekaj trenutkov pozneje pride do okvare. Trajnost zagotavlja, da zapisi okoljskega spremljanja ostanejo nedotaknjeni skozi leta analiziranja in poročanja.

Lastnosti ACID – atomarnost, konsistentnost, izolacija in trajnost – zagotavljajo, da je vsaka operacija nad podatki popolna, veljavna, brez medsebojnih motenj in trajna.

Primer uporabe

Za učenje jezika SQL bomo uporabljali spletno orodje **Oracle FreeSQL** (https://freesql.com/). **Oracle FreeSQL** študentom omogoča brezplačen dostop do resnične podatkovne zbirke **Oracle** v oblaku, kjer lahko v profesionalnem okolju vadijo SQL in koncepte relacijskih podatkovnih zbirk. Za delo v okolju **Oracle FreeSQL** se morate prijaviti v svoj račun (Sign in) ali se registrirati (Sign up), če računa še nimate.



Slika 5: Okolje Oracle FreeSQL

Pred vadbo jezika SQL je pomembno razumeti, kakšno podatkovno zbirko bomo zgradili in zakaj.

V tem predmetu bomo izdelali relacijsko učno podatkovno zbirko, ki predstavlja spremljanje kakovosti zraka v Varšavi na Poljskem. Podatkovna zbirka je namenjena izobraževanju in temelji na logiki dejanskih okoljskih informacijskih sistemov, ki jih uporabljajo javni organi in raziskovalne ustanove. Čeprav so vrednosti podatkov zaradi lažjega učenja poenostavljene, struktura odraža stvarna razmerja in omejitve.

Podatkovna zbirka predstavlja, kako merilne postaje zbirajo meritve onesnaženosti zraka, kako so te postaje opisane z metapodatki ter kako so vrste onesnaževal opredeljene in zakonsko urejene. Vsak od teh konceptov je v skladu z načeli relacijskega modela shranjen v ločeni tabeli.

Jedro podatkovne zbirke so merilne postaje, ki predstavljajo fizične lokacije v Varšavi, kjer so nameščeni senzorji kakovosti zraka. Vsaka postaja ima identifikator, ime, geografsko lokacijo in opisne atribute, kot sta regija in upravljavec. Postaje so shranjene neodvisno, zato je njihove metapodatke mogoče vzdrževati, tudi če meritve še niso na voljo.

Druga ključna sestavina podatkovne zbirke je šifrant onesnaževal. Onesnaževala, kot so dušikov dioksid (NO_2), trdni delci (PM10, PM2.5), ozon (O_3) in žveplov dioksid (SO_2), so opredeljena enkrat in se nato dosledno uporabljajo po vsej podatkovni zbirki. Tak pristop preprečuje podvajanje, zagotavlja celovitost podatkov in omogoča smiselne primerjave med meritvami.

Podatki meritev predstavljajo glavni vsebinski del podatkovne zbirke. Vsaka meritev beleži koncentracijo določenega onesnaževala na določeni postaji in ob določenem času. Meritve so z relacijskimi ključi povezane tako s postajami kot z onesnaževali, kar omogoča poizvedbe, ki v enem nizu rezultatov združujejo lokacijo, čas in vrsto onesnaževala.

Podatkovna zbirka poleg tega vsebuje tabelo z zakonsko določenimi mejnimi vrednostmi, ki predstavljajo poenostavljene evropske mejne vrednosti kakovosti zraka. Te vrednosti omogočajo analitične poizvedbe za odkrivanje preseganj in podpirajo primerjave med izmerjenimi koncentracijami ter sprejetimi standardi.

Celotna struktura upošteva načela normalizacije in referenčne celovitosti. Vsak koncept iz resničnega sveta predstavlja lastna tabela, razmerja med tabelami pa so izrecno določena. Takšna zasnova zagotavlja, da je podatkovna zbirka dosledna, razširljiva in primerna za analitične poizvedbe SQL, ki vključujejo združevanje tabel, agregiranje in podpoizvedbe.

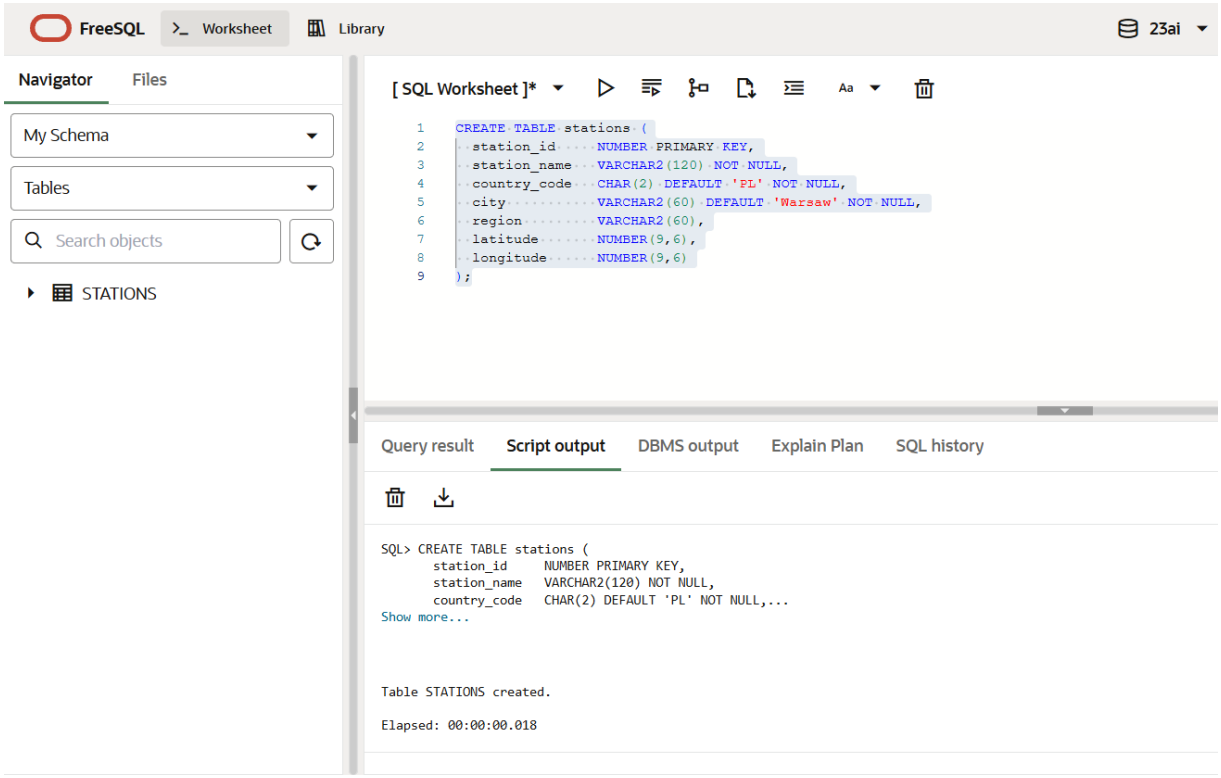
1. Operacije CREATE v jeziku SQL

Podatkovno zbirko bomo ustvarili s postopnim ustvarjanjem tabel.

1.1 Tabelo stations ustvarite tako, da kodo SQL vnesete v polje SQL Worksheet in nato pritisnete gumb **Run Statement**:

```
CREATE TABLE stations (
  station_id     NUMBER PRIMARY KEY,
  station_name   VARCHAR2(120) NOT NULL,
  country_code   CHAR(2) DEFAULT 'PL' NOT NULL,
  city           VARCHAR2(60) DEFAULT 'Warsaw' NOT NULL,
```

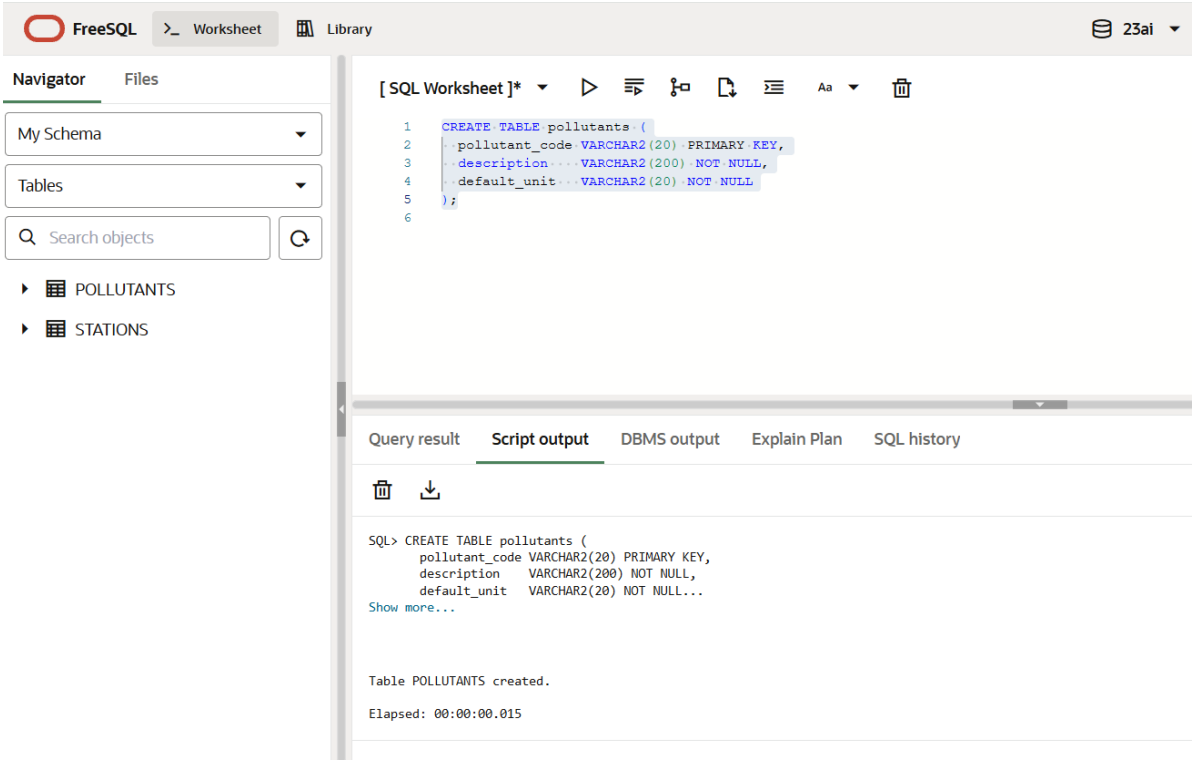
```
  region          VARCHAR2(60),
  latitude        NUMBER(9,6),
  longitude       NUMBER(9,6)
);
```



Slika 6: Ustvarjanje tabele stations

1.2 Ustvarjanje tabele pollutants

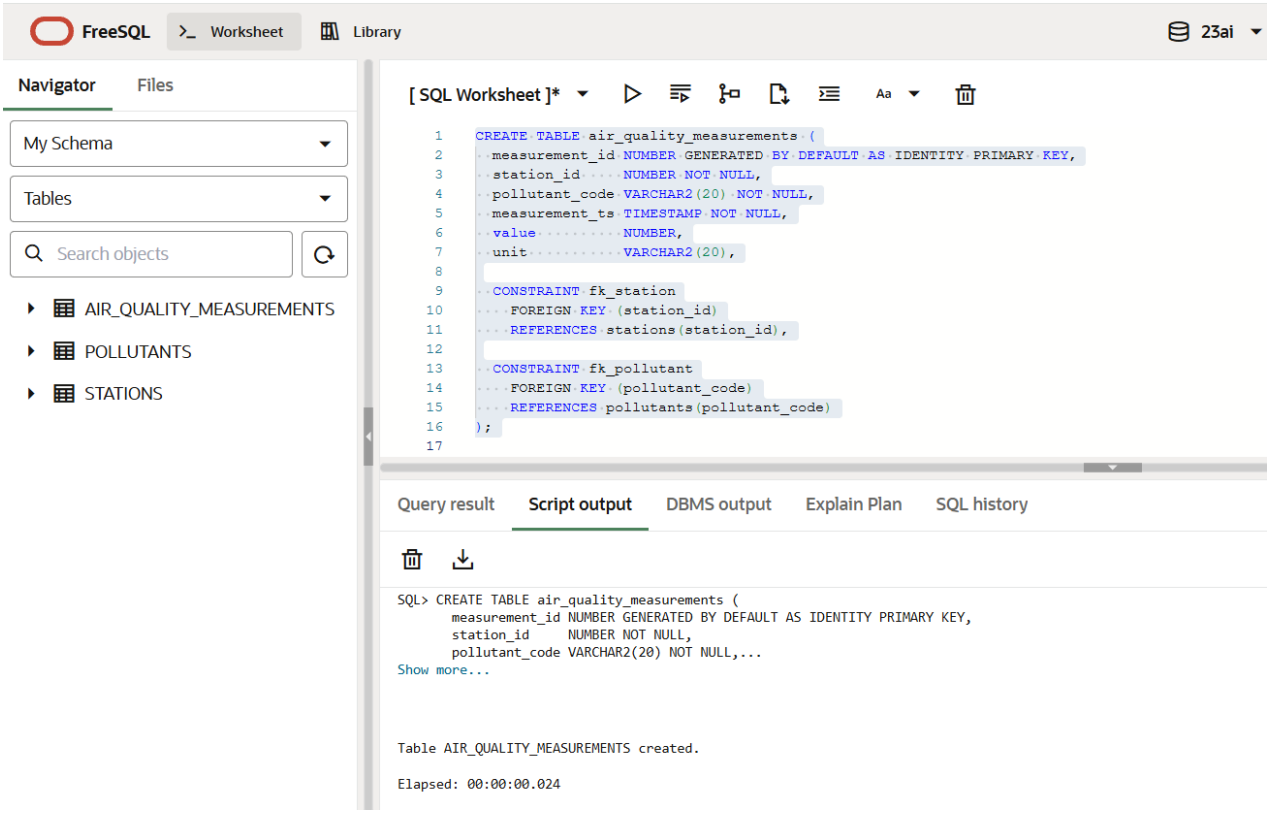
```
CREATE TABLE pollutants (
  pollutant_code VARCHAR2(20) PRIMARY KEY,
  description    VARCHAR2(200) NOT NULL,
  default_unit   VARCHAR2(20) NOT NULL
);
```



Slika 7: Ustvarjanje tabele pollutants

1.3 Ustvarjanje tabele meritev kakovosti zraka:

```
CREATE TABLE air_quality_measurements (
  measurement_id NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  station_id     NUMBER NOT NULL,
  pollutant_code VARCHAR2(20) NOT NULL,
  measurement_ts TIMESTAMP NOT NULL,
  value          NUMBER,
  unit           VARCHAR2(20),

  CONSTRAINT fk_station
    FOREIGN KEY (station_id)
    REFERENCES stations(station_id),

  CONSTRAINT fk_pollutant
    FOREIGN KEY (pollutant_code)
    REFERENCES pollutants(pollutant_code)
);
```



Slika 8: Ustvarjanje tabele meritev kakovosti zraka

Na tej stopnji je zbirka podatkov ustvarjena. Ali so se tabele pojavile, lahko vizualno preverite tako, da v podoknu Navigator na levi poiščete imena tabel STATIONS, POLLUTANTS in AIR_QUALITY_MEASUREMENTS.

2. Operacije INSERT

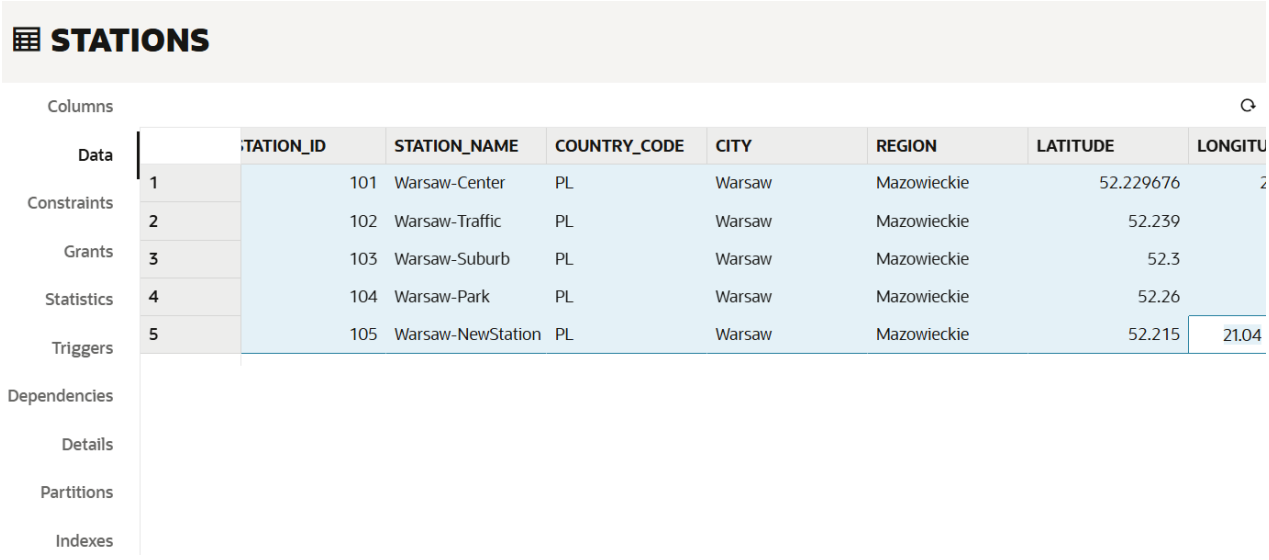
Zdaj napolnimo tabele s podatki.

2.1 Zdaj bomo vstavili podatke v tabelo stations:

```
INSERT INTO stations VALUES
(101, 'Warsaw-Center', 'PL', 'Warsaw', 'Mazowieckie', 52.229676, 21.012229);


INSERT INTO stations VALUES
(102, 'Warsaw-Traffic', 'PL', 'Warsaw', 'Mazowieckie', 52.239000, 21.005000);
```

```
INSERT INTO stations VALUES
(103, 'Warsaw-Suburb', 'PL', 'Warsaw', 'Mazowieckie', 52.300000, 20.900000);

INSERT INTO stations VALUES
(104, 'Warsaw-Park', 'PL', 'Warsaw', 'Mazowieckie', 52.260000, 21.020000);

-- Postaja brez meritev (za preizkušanje LEFT JOIN)
INSERT INTO stations VALUES
(105, 'Warsaw-NewStation', 'PL', 'Warsaw', 'Mazowieckie', 52.215000, 21.040000);
```

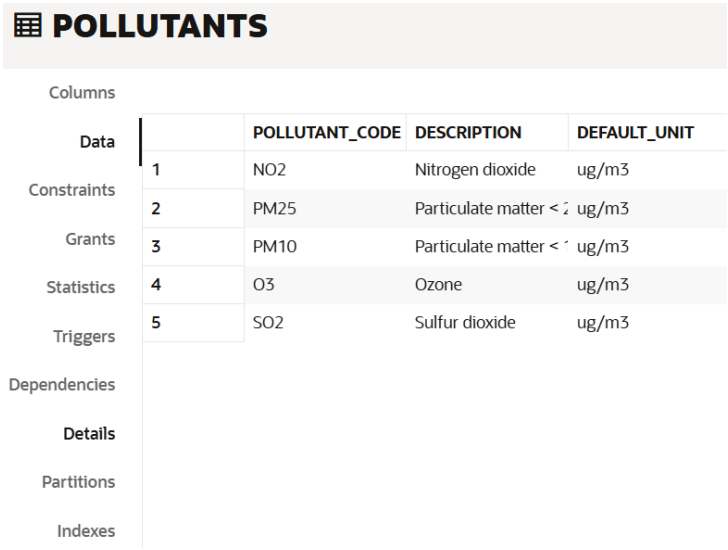
Po izvedbi ukaza lahko z desno miškino tipko kliknemo tabelo STATIONS v levem podoknu, jo odpremo in preverimo, da je napolnjena s podatki (slika 9).



Slika 9: Rezultat vnosa podatkov v tabelo stations

2.2 Vstavimo podatke o onesnaževalih:

```
INSERT INTO pollutants VALUES
('NO2', 'Nitrogen dioxide', 'ug/m3');

INSERT INTO pollutants VALUES
('PM25', 'Particulate matter < 2.5 µm', 'ug/m3');

INSERT INTO pollutants VALUES
('PM10', 'Particulate matter < 10 µm', 'ug/m3');

INSERT INTO pollutants VALUES
('O3', 'Ozone', 'ug/m3');

-- Onesnaževalo brez meritev (za preizkušanje RIGHT JOIN)
INSERT INTO pollutants VALUES
('SO2', 'Sulfur dioxide', 'ug/m3');
```



POLLUTANTS

Columns | Data | Constraints | Grants | Statistics | Triggers | Dependencies | Details | Partitions | Indexes

	POLLUTANT_CODE	DESCRIPTION	DEFAULT_UNIT
1	NO2	Nitrogen dioxide	ug/m3
2	PM25	Particulate matter < 2	ug/m3
3	PM10	Particulate matter < 1	ug/m3
4	O3	Ozone	ug/m3
5	SO2	Sulfur dioxide	ug/m3

Slika 10: Rezultat vnosa podatkov o onesnaževalih

2.3 Zdaj vstavimo podatke o meritvah:

```
INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
```

```
(101, 'NO2', TIMESTAMP '2025-03-01 08:00:00', 45, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(102, 'PM25', TIMESTAMP '2025-03-01 08:00:00', 28, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(101, 'NO2', TIMESTAMP '2025-03-01 09:00:00', 78, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(102, 'NO2', TIMESTAMP '2025-03-01 08:00:00', 210, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(103, 'PM25', TIMESTAMP '2025-03-01 08:00:00', 18, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(104, 'PM10', TIMESTAMP '2025-03-01 08:00:00', 62, 'ug/m3');

INSERT INTO air_quality_measurements
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(104, 'O3', TIMESTAMP '2025-03-01 13:00:00', 130, 'ug/m3');

-- Napačna meritev (za preizkušanje ukazov UPDATE in DELETE)
INSERT INTO air_quality_measurements
```

```
(station_id, pollutant_code, measurement_ts, value, unit)
VALUES
(101, 'PM25', TIMESTAMP '2025-03-01 10:00:00', -7, 'ug/m3');
```

AIR_QUALITY_MEASUREMENTS

Columns | Data | Constraints | Grants | Statistics | Triggers | Dependencies | Details | Partitions | Indexes

	MEASUREMENT_ID	STATION_ID	POLLUTANT_CODE	MEASUREMENT_TS	VALUE	UNIT
1	1	101	NO2	2025-03-01T08:00:(	45	ug/m3
2	2	102	PM25	2025-03-01T08:00:(	28	ug/m3
3	3	101	NO2	2025-03-01T09:00:(	78	ug/m3
4	4	102	NO2	2025-03-01T08:00:(	210	ug/m3
5	5	103	PM25	2025-03-01T08:00:(	18	ug/m3
6	6	104	PM10	2025-03-01T08:00:(	62	ug/m3
7	7	104	O3	2025-03-01T13:00:(	130	ug/m3
8	8	101	PM25	2025-03-01T10:00:(	-7	ug/m3

Slika 11: Rezultat vnosa podatkov o meritvah

3. Operaciji UPDATE in DELETE

3.2 Z ukazom UPDATE lahko popravimo napačne vrednosti meritev tako, da negativne vrednosti zamenjamo z NULL, nato pa z ukazom DELETE odstranimo vse napačne vnose:

```
UPDATE air_quality_measurements
SET value = NULL
WHERE value < 0;
DELETE FROM air_quality_measurements
WHERE value IS NULL;
```

AIR_QUALITY_MEASUREMENTS

Columns | Data | Constraints | Grants | Statistics | Triggers | Dependencies | Details

	MEASUREMENT_ID	STATION_ID	POLLUTANT_CODE	MEASUREMENT_TS	VALUE	UNIT
1	1	101	NO2	2025-03-01T08:00:(	45	ug/m3
2	2	102	PM25	2025-03-01T08:00:(	28	ug/m3
3	3	101	NO2	2025-03-01T09:00:(	78	ug/m3
4	4	102	NO2	2025-03-01T08:00:(	210	ug/m3
5	5	103	PM25	2025-03-01T08:00:(	18	ug/m3
6	6	104	PM10	2025-03-01T08:00:(	62	ug/m3
7	7	104	O3	2025-03-01T13:00:(	130	ug/m3

Slika 12: Rezultat po odstranitvi napačnih vrednosti meritev

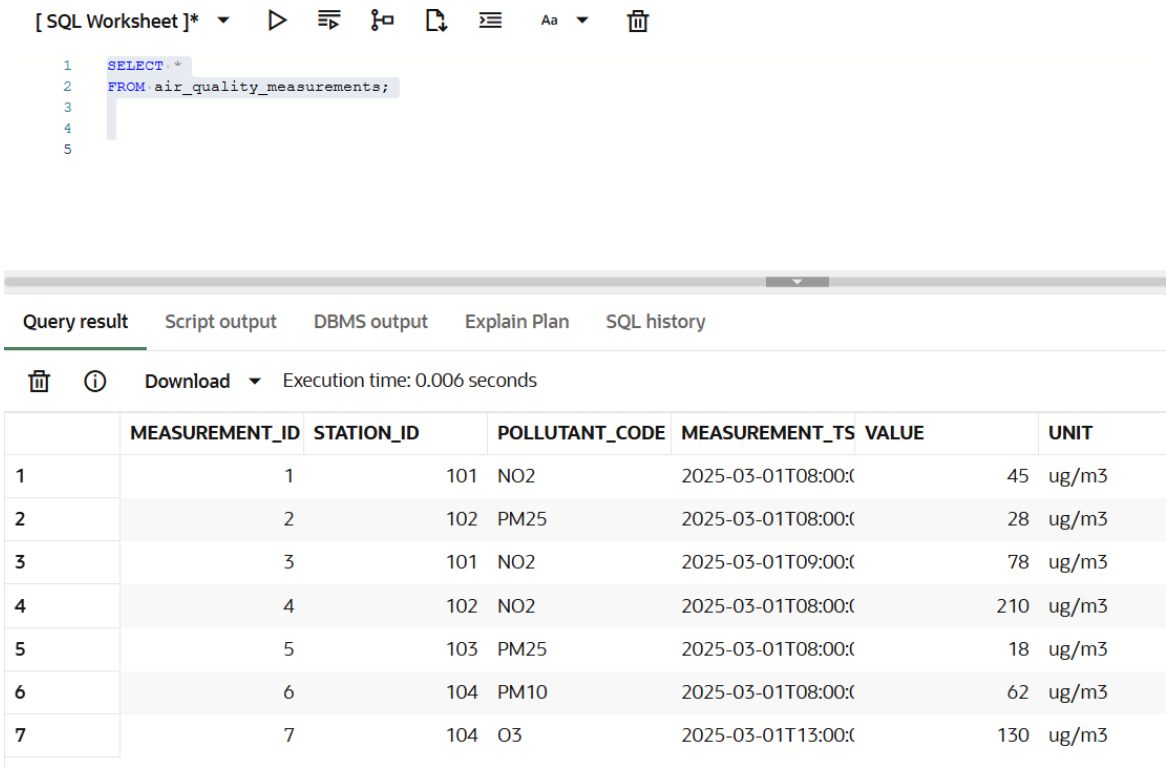
3.2 Napišite podoben ukaz za odstranjevanje postaj, pri katerih je vrednost regije NULL.

4. Poizvedbe SELECT

Spoznali smo ukaze DML, zato zdaj preizkusimo še ukaz za poizvedbo SELECT.

4.1 Preprosta poizvedba SELECT:

```
SELECT *
FROM air_quality_measurements;
```



	MEASUREMENT_ID	STATION_ID	POLLUTANT_CODE	MEASUREMENT_TS	VALUE	UNIT
1	1	101	NO2	2025-03-01T08:00:(	45	ug/m3
2	2	102	PM25	2025-03-01T08:00:(	28	ug/m3
3	3	101	NO2	2025-03-01T09:00:(	78	ug/m3
4	4	102	NO2	2025-03-01T08:00:(	210	ug/m3
5	5	103	PM25	2025-03-01T08:00:(	18	ug/m3
6	6	104	PM10	2025-03-01T08:00:(	62	ug/m3
7	7	104	O3	2025-03-01T13:00:(	130	ug/m3

Slika 13: Rezultat izvedbe poizvedbe

4.2 Zdaj v praksi preverite, kako se bo rezultat spremenil ob izvedbi naslednje poizvedbe, ki rezultate razvrsti s stavkom ORDER BY:

```
SELECT station_id, pollutant_code, value
FROM air_quality_measurements
ORDER BY value DESC;
```

4.3 Zdaj uporabimo filtriranje WHERE z logičnimi operatorji, da pridobimo podatke samo za izbrano onesnaževalo:

```
SELECT station_id, pollutant_code, value
FROM air_quality_measurements
WHERE value > 50
  AND pollutant_code = 'NO2';
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.007 seconds

	STATION_ID	POLLUTANT_CODE	VALUE
1	101	NO2	78
2	102	NO2	210

Slika 14: Rezultat izvedbe poizvedbe

4.4 Poskusimo uporabiti agregatne funkcije z združevanjem. Kako se bo spremenil izpis?

```
SELECT station_id, pollutant_code, AVG(value) AS avg_value
FROM air_quality_measurements
WHERE value IS NOT NULL
GROUP BY station_id, pollutant_code;
```

4.5 Za filtriranje kombinacij postaj in onesnaževal s povprečnimi vrednostmi nad 40 poskusimo uporabiti agregatno funkcijo s stavkom HAVING:

```
SELECT station_id, pollutant_code, AVG(value) AS avg_value
FROM air_quality_measurements
WHERE value IS NOT NULL
GROUP BY station_id, pollutant_code
HAVING AVG(value) > 40;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.073 seconds

	STATION_ID	POLLUTANT_CODE	AVG_VALUE
1	101	NO2	61.5
2	102	NO2	210
3	104	PM10	62
4	104	O3	130

Slika 15: Rezultat izvedbe poizvedbe

5. Operacije JOIN

V relacijski zbirki podatkov so informacije le redko shranjene v eni sami tabeli. Podatki so namesto tega razporejeni med več tabelami, pri čemer vsaka predstavlja določen pojem iz resničnega sveta. Za učinkovito analizo takih podatkov je pogosto treba združiti vrstice iz različnih tabel v enoten nabor rezultatov. To dosežemo z operacijo **JOIN** v jeziku SQL.

JOIN določa, kako so vrstice dveh tabel povezane na podlagi logičnega pogoja, ki navadno vključuje primarni ključ ene tabele in tuji ključ druge. V učni zbirki podatkov, uporabljeni pri tem predmetu, so meritve kakovosti zraka povezane z merilnimi postajami prek atributa station_id. Ta povezava omogoča, da poizvedbe hkrati pridobijo vrednosti meritev in opisne podatke o postajah.

5.1 INNER JOIN vrne samo tiste vrstice, za katere v obeh tabelah obstajajo ujemajoče se vrednosti. Vrstice brez ustreznega ujemanja v eni ali drugi tabeli so izključene iz rezultata.

V okviru zbirke podatkov o kakovosti zraka INNER JOIN med tabelo meritev in tabelo postaj vrne samo meritve, povezane z znano merilno postajo. To je najpogosteje uporabljena vrsta združevanja in je primerna, kadar želimo nepopolne ali nepovezane podatke zanemariti.

Pojmovno INNER JOIN odgovarja na vprašanje: »Kateri zapisi hkrati obstajajo v obeh tabelah?«

[SQL Worksheet]*

```
SELECT s.station_name, m.pollutant_code, m.value
FROM air_quality_measurements m
INNER JOIN stations s
  ON s.station_id = m.station_id;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.118 seconds

	STATION_NAME	POLLUTANT_CODE	VALUE
1	Warsaw-Center	NO2	45
2	Warsaw-Traffic	PM25	28
3	Warsaw-Center	NO2	78
4	Warsaw-Traffic	NO2	210
5	Warsaw-Suburb	PM25	18
6	Warsaw-Park	PM10	62
7	Warsaw-Park	O3	130

Slika 16: Rezultat poizvedbe INNER JOIN

Če v poizvedbi vidite samo besedo JOIN, ta prav tako pomeni INNER JOIN (slika 17).

[SQL Worksheet]*

```
SELECT
  s.station_name,
  m.pollutant_code,
  m.value,
  m.measurement_ts
FROM air_quality_measurements m
JOIN stations s
  ON s.station_id = m.station_id;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.006 seconds

	STATION_NAME	POLLUTANT_CODE	VALUE	MEASUREMENT_TS
1	Warsaw-Center	NO2	45	2025-03-01T08:00:00Z
2	Warsaw-Traffic	PM25	28	2025-03-01T08:00:00Z
3	Warsaw-Center	NO2	78	2025-03-01T09:00:00Z
4	Warsaw-Traffic	NO2	210	2025-03-01T08:00:00Z
5	Warsaw-Suburb	PM25	18	2025-03-01T08:00:00Z
6	Warsaw-Park	PM10	62	2025-03-01T08:00:00Z
7	Warsaw-Park	O3	130	2025-03-01T13:00:00Z

Slika 17: Rezultat poizvedbe INNER JOIN

5.2 LEFT JOIN vrne vse vrstice iz leve tabele, tudi če v desni tabeli ni ujemajoče se vrstice. Kadar ujemanja ni, stolpci iz desne tabele vsebujejo vrednosti NULL.

Ta vrsta združevanja je še posebej uporabna za odkrivanje manjkajočih ali nepopolnih podatkov. Pri združevanju merilnih postaj z meritvami kakovosti zraka denimo LEFT JOIN omogoča, da poizvedba navede vse postaje, tudi tiste, za katere še ni zabeleženih meritev.

LEFT JOIN odgovarja na vprašanje: »Kateri zapisi obstajajo v levi tabeli in kateri povezani podatki (če sploh kateri) so na voljo v desni tabeli?«

[SQL Worksheet]*

```
SELECT s.station_name, m.pollutant_code, m.value
FROM stations s
LEFT JOIN air_quality_measurements m
  ON s.station_id = m.station_id
ORDER BY s.station_name;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.073 seconds

	STATION_NAME	POLLUTANT_CODE	VALUE
1	Warsaw-Center	NO2	45
2	Warsaw-Center	NO2	78
3	Warsaw-NewStation	*(null)*	*(null)*
4	Warsaw-Park	O3	130
5	Warsaw-Park	PM10	62
6	Warsaw-Suburb	PM25	18
7	Warsaw-Traffic	PM25	28
8	Warsaw-Traffic	NO2	210

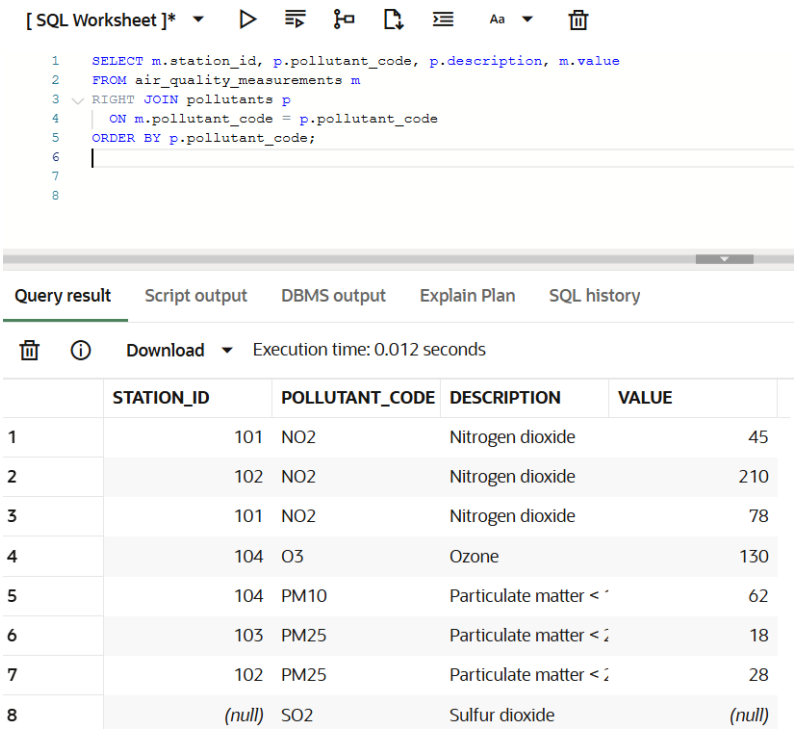
Slika 18: Rezultat poizvedbe LEFT JOIN

Kot lahko vidite, postaja Warsaw-NewStation nima povezanih meritev, vendar je kljub temu prikazana v rezultatih.

5.3 RIGHT JOIN deluje v nasprotni smeri. Vrne vse vrstice iz desne tabele, tudi če v levi tabeli ni ujemajoče se vrstice. Če ujemanja ni, so stolpci iz leve tabele zapolnjeni z vrednostmi NULL.

V učni zbirki podatkov lahko RIGHT JOIN uporabimo za prikaz vseh opredeljenih onesnaževal, tudi tistih, za katera še ni zabeleženih meritev. To je uporabno pri preverjanju popolnosti referenčnih podatkov ali prepoznavanju neuporabljenih kategorij.

RIGHT JOIN odgovarja na vprašanje: »Kateri zapisi obstajajo v desni tabeli in ali se nanje trenutno sklicuje leva tabela?«



Slika 19: Rezultat poizvedbe RIGHT JOIN

Kot lahko vidite, je v zadnji vrstici rezultatov RIGHT JOIN prikazano onesnaževalo SO2, ki ni povezano z nobeno postajo.

6. Podpoizvedbe v jeziku SQL

Podpoizvedba je poizvedba, ugnezdena v drug stavek SQL. Uporabimo jo, kadar je rezultat ene poizvedbe potreben za določitev rezultata druge. Podpoizvedbe omogočajo, da stavki SQL izrazijo zapleteno logiko na jasen in strukturiran način.

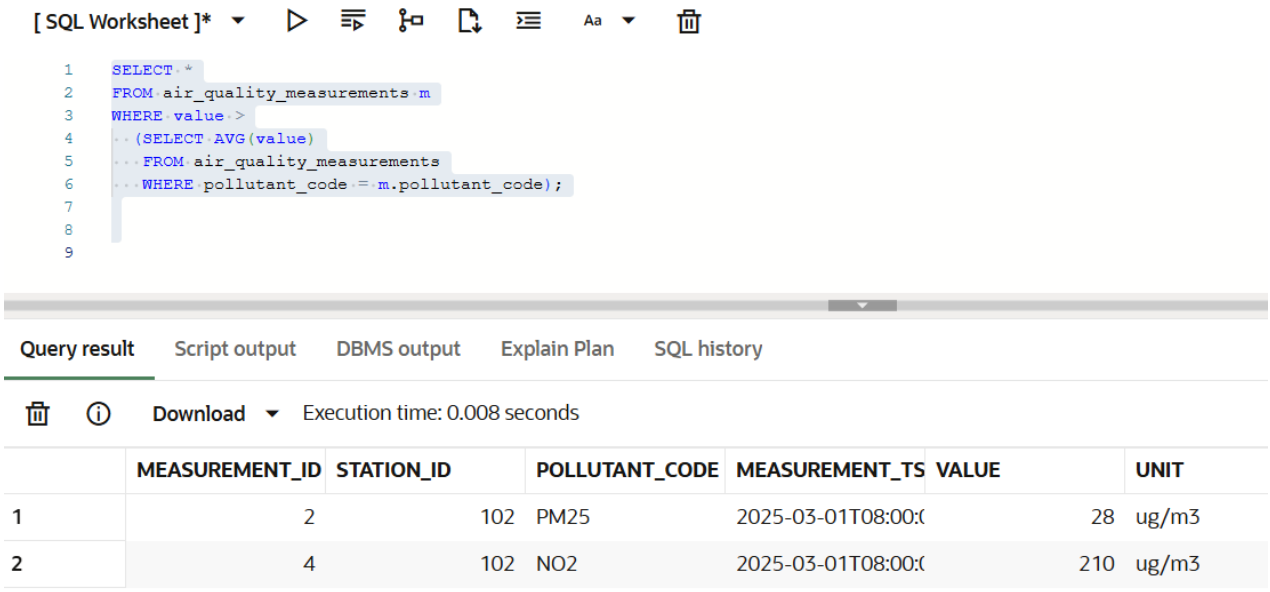
Podpoizvedba se običajno izvede najprej, njen rezultat pa nato uporabi zunanja poizvedba. Podpoizvedbe so še posebej uporabne pri analitičnih nalogah, ki vključujejo izračunane vrednosti, kot so povprečja, maksimumi, minimumi ali dinamično določeni pragovi.

Podpoizvedbe lahko glede na svoj namen vrnejo eno vrednost, eno vrstico ali množico vrstic. Pogost primer uporabe je primerjava posameznih zapisov z agregiranimi vrednostmi, izpeljanimi iz iste tabele.

Podpoizvedbe so lahko nekorelirane, kar pomeni, da se izvajajo neodvisno od zunanje poizvedbe, ali korelirane, kar pomeni, da so odvisne od vrednosti iz zunanje poizvedbe in se

ovrednotijo znova za vsako vrstico. Čeprav so korelirane podpoizvedbe zahtevnejše, imata obe vrsti pomembno vlogo pri analizi relacijskih podatkov.

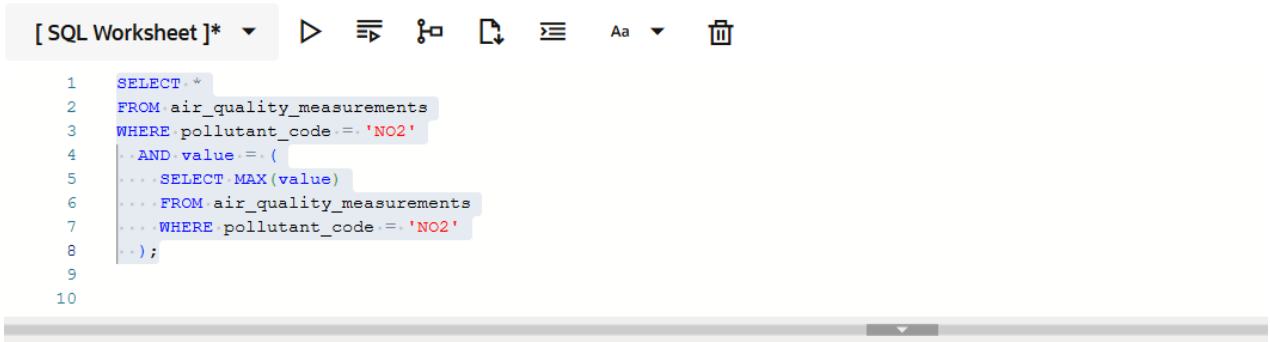
6.1 Ustvarimo podpoizvedbo, ki bo prikazala meritve, višje od povprečne vrednosti za določeno onesnaževalo (slika 20).



	MEASUREMENT_ID	STATION_ID	POLLUTANT_CODE	MEASUREMENT_TS	VALUE	UNIT
1	2	102	PM25	2025-03-01T08:00:(	28	ug/m3
2	4	102	NO2	2025-03-01T08:00:(	210	ug/m3

Slika 20: Rezultat izvedbe podpoizvedbe

6.2 Podpoizvedba nam lahko pomaga tudi prepoznati postaje z največjimi vrednostmi NO_2 (slika 21).



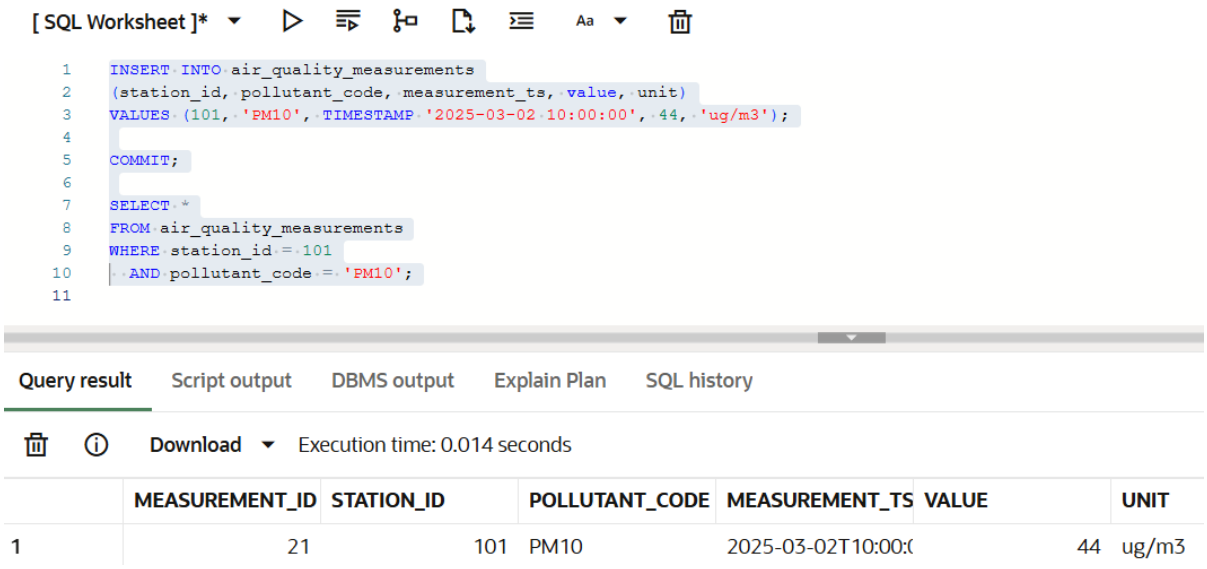
	MEASUREMENT_ID	STATION_ID	POLLUTANT_CODE	MEASUREMENT_TS	VALUE	UNIT
1	4	102	NO2	2025-03-01T08:00:(	210	ug/m3

Slika 21: Rezultat izvedbe podpoizvedbe

7. Jezik za nadzor transakcij (TCL)

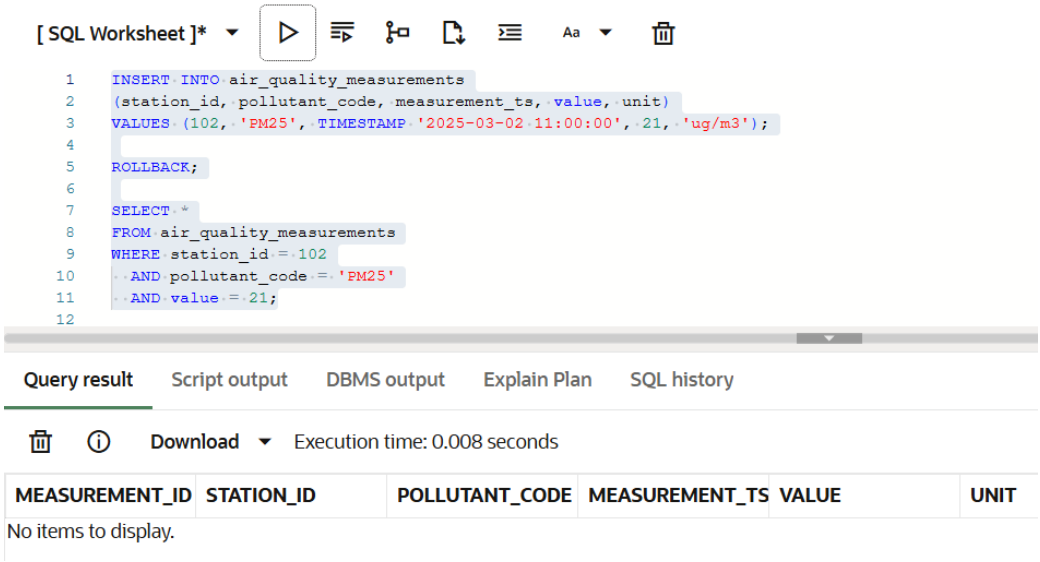
Jezik za nadzor transakcij (TCL) zagotavlja zanesljivo izvajanje skupin operacij SQL. Transakcije se v okolju Oracle začnejo samodejno, vendar jih je treba izrecno dokončati ali preklicati z ukazom **COMMIT** oziroma **ROLLBACK**. Ukaz **SAVEPOINT** omogoča delno razveljavitev znotraj transakcije.

7.1 Vstavite novo meritev kakovosti zraka in trajno shranite spremembo.



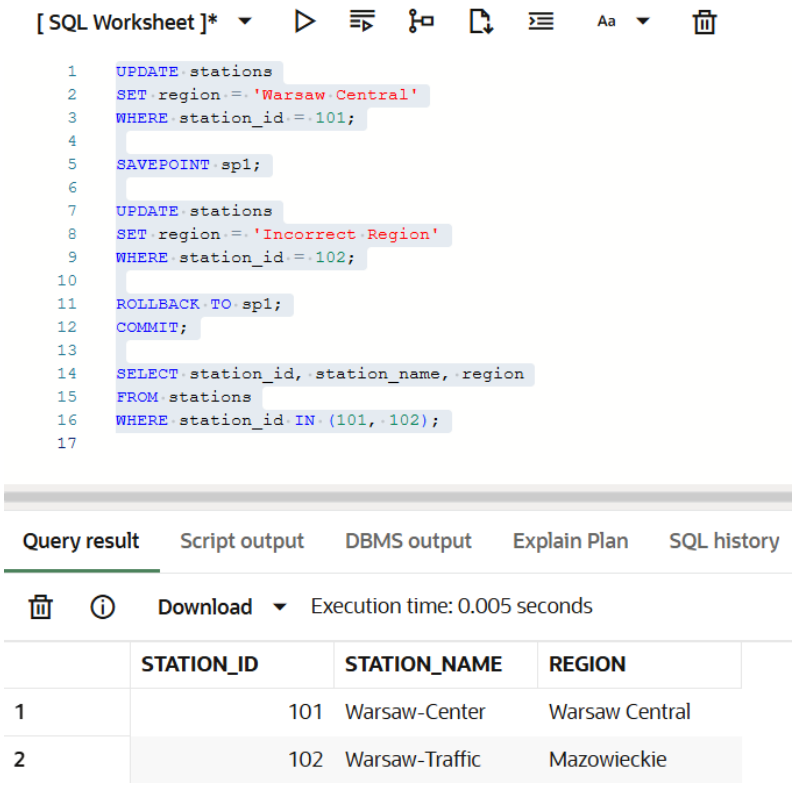
Slika 22: Preverjanje potrjene transakcije

7.2 Vstavite meritev in nato prekličite operacijo z ukazom ROLLBACK.



Slika 23: Preverjanje razveljavitve transakcije

7.3 Izvedite več posodobitev in razveljavite samo del transakcije.



Slika 24: Preverjanje delne razveljavitve transakcije

Analitične naloge SQL:

1. Prikažite vse meritve kakovosti zraka, vključno z identifikatorjem postaje, kodo onesnaževala, izmerjeno vrednostjo in časovnim žigom. Rezultate razvrstite po času meritve.
2. Navedite vse meritve, pri katerih je zabeležena vrednost večja od 50. Prikažite identifikator postaje, onesnaževalo, vrednost in čas.
3. Pridobite vse meritve NO_2, razvrščene od najvišje do najnižje vrednosti.
4. Za vsako onesnaževalo poiščite postajo, na kateri je bila zabeležena največja izmerjena vrednost. Prikažite ime postaje, onesnaževalo in izmerjeno vrednost.
5. Prikažite samo tiste kombinacije postaj in onesnaževal, pri katerih povprečna vrednost presega 40.
6. Za vsako onesnaževalo poiščite največjo zabeleženo vrednost meritve.

Praktične vaje

Vaja 1 – Ustvarjanje nove sheme zbirke podatkov

Za analizo podatkovnega nabora India Urban Noise Monitoring (na voljo na https://www.kaggle.com/datasets/rohanrao/noise-monitoring-data-in-india) morajo študenti vzpostaviti shemo zbirke podatkov z uporabo stavkov DDL. Ustvarite dve relacijski tabeli: STATIONS (Station, Name, City, State, Type) in STATION_MONTH (Station, Year, Month, Day, Night, DayLimit, NightLimit). Polje Station določite kot primarni ključ tabele STATIONS. Nato v tabeli STATION_MONTH določite tuji ključ, tako da se STATION_MONTH.Station sklicuje na STATIONS(Station), s čimer zagotovite referenčno celovitost med zapisi meritev in metapodatki postaj.

Vaja 2 – Vstavljanje vzorčnih podatkov

Ker Oracle FreeSQL ne podpira neposrednega uvoza datotek CSV in je v tem okolju osnovni način nalaganja podatkov stavek INSERT, je bil podatkovni nabor pretvorjen v dve skripti SQL: *stations_insert.sql* in *station_month_insert.sql*. Za polnjenje ustreznih tabel izvedite obe skripti. Najprej vstavite metapodatke postaj (STATIONS), nato pa naložite mesečne meritve (STATION_MONTH).

Vaja 3 – Osnovne poizvedbe

Napišite osnovne poizvedbe SELECT za raziskovanje podatkov. Najprej navedite vse postaje v izbranem mestu (na primer Delhi). Nato prikažite mesečne zapise meritev hrupa za izbrano postajo in leto. Te poizvedbe naj študentom pomagajo razumeti zgradbo in pomen obeh tabel pred prehodom na analitične naloge.

Vaja 4 – Filtriranje z logičnimi operatorji

Pridobite in primerjajte dve kategoriji primerov prekoračitev. Najprej vrnite vse zapise, pri katerih sta preseženi obe mejni vrednosti, kar pomeni Day > DayLimit AND Night > NightLimit. Nato vrnite vse zapise, pri katerih je presežena vsaj ena mejna vrednost, kar pomeni Day > DayLimit OR Night > NightLimit. Razložite, kako izbira logičnega operatorja spremeni pomen in velikost nabora rezultatov.

Vaja 5 – Agregacije

Izračunajte povzetek po postajah, tako da za vsako postajo določite povprečno dnevno in povprečno nočno raven hrupa. Rezultate združite po polju Station. Ta vaja utrjuje uporabo agregatnih funkcij in stavka GROUP BY pri analitičnem poročanju.

Vaja 5 – Poizvedbe JOIN

Pripravite združeno poročilo, ki povezuje metapodatke postaj z mesečnimi podatki meritev. Z uporabo INNER JOIN med tabelama STATIONS in STATION_MONTH ustvarite nabor rezultatov, ki vključuje ime postaje, mesto, vrsto postaje, leto, mesec ter dnevno in nočno raven hrupa. Ta vaja prikazuje, kako relacijske povezave omogočajo smiselna poročila, ki združujejo opisne in izmerjene podatke.

Ocenjevanje

Kviz (5 vprašanj): kratko preverjanje pojmov pred praktičnim delom.

1. Katera trditev najbolje opisuje LEFT JOIN?

A. Vrne samo vrstice, ki se ujemajo v obeh tabelah

B. Vrne vse vrstice iz leve tabele in ujemajoče se vrstice iz desne tabele

C. Vrne vse vrstice iz desne tabele in ujemajoče se vrstice iz leve tabele

D. Brez pogojev združi vsako vrstico ene tabele z vsako vrstico druge tabele

2. Katera vrsta anomalije nastane, kadar se ob izbrisu zadnje meritve za postajo izgubijo tudi metapodatki te postaje?

A. Anomalija posodabljanja

B. Anomalija vstavljanja

C. Anomalija brisanja

D. Anomalija združevanja

3. Kateri ukaz SQL spada v jezik za opredelitev podatkov (DDL)?

A. SELECT

B. INSERT

C. DROP

D. UPDATE

4. Kaj od navedenega je veljaven primer uporabe podpoizvedbe v jeziku SQL?

A. Izračun povprečne vrednosti onesnaževala znotraj stavka WHERE

B. Brisanje zbirke podatkov

C. Dodeljevanje dovoljenj

D. Ustvarjanje novih tabel

5. Kateri stavek se uporablja za združevanje vrstic iz dveh tabel na podlagi pogoja ujemanja?

A. GROUP BY

B. ORDER BY

C. JOIN … ON

D. HAVING

Merila ocenjevanja: pravilnost poizvedb SQL, popolnost rezultatov, jasnost razlage in rezultat kviza.

Element	Delež	Merila
Kviz	20%	Razumevanje pojmov
Naloge SQL	50%	Pravilnost, učinkovitost, celovitost
Poročilo	30%	Razlaga, jasnost, vizualizacija

Viri in dodatno branje

Viri Oracle Academy Cloud

Dokumentacija o standardih EU za kakovost zraka

Referenčna dokumentacija API-ja Open-Meteo

Izbrane prosto dostopne knjige o podatkovnem skladiščenju

Izbrani prosto dostopni raziskovalni članki o podatkovnem skladiščenju v okoljski znanosti

C. J. Date. Introduction to Database Systems. Addison-Wesley Longman, Inc., 2004. 983 p.

H. Garcia-Molina, J. Ullman, J. Widom. Database Systems: The Complete Book. Pearson, 2008. 1248 p.

P. Rob, C. Coronel. Database Systems: Design, Implementation, and Management. Course Technology, 2011. 728 p.