SADRŽAJ

UVOD

Baze podataka

Baza podataka je organizovana kolekcija podataka koja se čuva na računaru ili drugom uređaju. Podaci u bazi podataka su organizovani na način koji omogućava brz i efikasan pristup i upravljanje informacijama. Baze podataka se koriste u različitim aplikacijama, od jednostavnih aplikacija kao što su programi za vođenje evidencije o klijentima, do složenih aplikacija kao što su bankarski sistemi ili globalni sistemi za upravljanje ljudskim resursima.

Baze podataka su dizajnirane tako da omogućavaju korisnicima da lako pretražuju i ažuriraju podatke u sistemu, bez potrebe da brinu o tome gde se podaci nalaze ili kako su organizovani. Baze podataka su takođe dizajnirane da obezbede sigurnost i integritet podataka, tako da se spreči gubitak podataka ili neovlašćen pristup informacijama.

Postoji nekoliko vrsta baza podataka, kao što su relacione baze podataka, hijerarhijske baze podataka, mrežne baze podataka i objektno-orijentisane baze podataka. Relacione baze podataka su najčešće korišćene i koriste se u različitim aplikacijama, uključujući računovodstvo, upravljanje inventarom i upravljanje projektima.

SQL

SQL (Structured Query Language) je standardizovan jezik za upravljanje i manipulisanje podacima u bazi podataka. Koristi se za kreiranje, ažuriranje, brisanje i izdvajanje podataka iz baze podataka. SQL se koristi u većini vrsta baza podataka, uključujući relacijske baze podataka, objektno-orijentisane baze podataka i hijerarhijske baze podataka. SQL omogućava programerima, administratorima baze podataka i drugim korisnicima da efikasno i pouzdano izvršavaju upite nad bazama podataka, čime se omogućava efikasno skladištenje, organizovanje i pristupanje podacima. SQL je jedan od najpopularnijih jezika za rad sa bazama podataka i široko se koristi u industriji i akademskoj zajednici.

ENTITET

Entitet u bazama podataka predstavlja nešto što ima jedinstveno ime i svojstva koja se mogu opisati. To može biti nešto kao što su osoba, proizvod ili mesto. Entiteti su osnovni elementi baze podataka i koriste se za organizovanje podataka na način koji olakšava njihovu pretragu, izmenu i upotrebu.

Svaki entitet ima svoja svojstva koja se nazivaju atributi. Atributi opisuju karakteristike entiteta i sadrže informacije koje su povezane sa entitetom. Na primer, atributi osobe mogu uključivati ime, prezime, datum rođenja, adresu, broj telefona i slično.

Entiteti se koriste za stvaranje tabela u bazi podataka. Svaki entitet predstavlja jednu tabelu, a svaki atribut predstavlja jednu kolonu u tabeli. Svaki red u tabeli predstavlja jedan primerak entiteta.

Primer ispravno kreiranog entiteta : Učenik (JMBG, Ime, Prezime, DatumRođenja)

Primer neispravno kreiranog entiteta : Učenik (JMBG, Ime, Prezime, DatumRođenja, ImeŠkole, MestoŠkole, AdresaŠkole)

Neispravni su oni entiteti koji u sebi sadrže atribute od kojih možemo sastaviti poseban entitet (u gornjem slučaju ImeŠkole, MestoŠkole, AdresaŠkole se mogu pretvoriti u poseban entitet Škola).

Pitanja iz priručnika za maturski ispit

208. Међу понуђеним ентитетима, одредити ентитет са атрибутима који **НИСУ** одговарајући. Заокружити број испред траженог одговора:

1. Ентитет: СТУДЕНТ – Атрибути: име, презиме, смер, број бодова, просек
2. Ентитет: ДРЖАВА – Атрибути: назив, број становника, површина
3. Ентитет: КЊИГА – Атрибути: наслов, аутор, година издања, издавач, адреса издавача, телефон издавача
4. Ентитет: АВИОН – Атрибути: произвођач, марка, година производње, број седишта

231 Заокружити бројеве испред тражених одговора.

Одредити ентитете који садрже одговарајуће атрибуте:

1. Ентитет: КЊИГА – Атрибути: наслов, аутор, издавач, година издања
2. Ентитет: АУТОМОБИЛ – Атрибути: марка, година производње, боја, власник, година рођења власника, регистарски број
3. Ентитет: УЧЕНИК – Атрибути: име, презиме, разред, одељење, број оправданих, број неоправданих, просек
4. Ентитет: ДРЖАВА – Атрибути: назив, број становника, површина, главни град, број становника главног града, име градоначелника главног града

243 Одредити ентитете који садрже одговарајуће атрибуте. Заокружити бројеве испред тражених одговора:

1. Ентитет: СТУДЕНТ – Атрибути: име, презиме, смер, број бодова, просек
2. Ентитет: КЊИГА – Атрибути: наслов, аутор, година издања, издавач, адреса издавача, телефон издавача
3. Ентитет: АВИОН – Атрибути: произвођач, марка, година производње, број седишта
4. Ентитет: ДРЖАВА – Атрибути: назив, број становника, површина
5. Ентитет: САЈАМ – Атрибути: назив, број излагача, покровитељ, адреса покровитеља, контак особа покровитеља
6. Ентитет: ТУРИСТИЧКА АГЕНЦИЈА – Атрибути: назив, адреса, година оснивања, власник, стручна квалификација власника, запослени, стручна квалификација запослених

KARDINALNOST

Kardinalnost u bazama podataka se odnosi na odnos između dve povezane tabele i određuje koliko puta se jedan red u jednoj tabeli može povezati sa redom u drugoj tabeli. Kardinalnost se obično prikazuje pomoću simbola na dijagramu ER (entitetsko-relacijski dijagram) ili korišćenjem brojeva u zagradama.

Postoje tri glavne vrste kardinalnosti:

Kardinalnost **1:1** (jedan prema jedan) - Ovaj odnos znači da se svaki red u jednoj tabeli povezuje sa tačno jednim redom u drugoj tabeli i obrnuto. Primer za to bi bio ako svaka osoba ima samo jedan pasoš određene države i svaki pasoš se povezuje sa tačno jednom osobom.

Kardinalnost **1:N** (jedan prema više) - Ovaj odnos znači da se svaki red u jednoj tabeli može povezati s više redova u drugoj tabeli, ali svaki red u drugoj tabeli se može povezati samo s jednim redom u prvoj tabeli. Primer za to bi bio ako svaka kategorija ima mnoge proizvode, ali svaki proizvod pripada samo jednoj kategoriji.

Kardinalnost **N:M** (više prema više) - Ovaj odnos znači da se više redova u jednoj tabeli može povezati s više redova u drugoj tabeli. Primer za to bi bio ako više profesora predaje više predmeta dok istovremeno svaki od tih predmeta predaje više profesora.

Pitanja iz priručnika za maturski ispit

214. Заокружити тачан исказ:

1. Кардиналност неке везе представља однос броја објеката који се повезују.
2. Кардиналност неке везе представља апстракцију у којој се скуп сличних типова објеката представља општим генеричким типом (надтипом).
3. Кардиналност неке везе одређује опционалност учешћа у вези.
4. Кардиналност показује колико кандидата за примарни кључ има неки тип ентитета.

252. Исписати на цртама испред релација редни број под којим је наведена одговарајућа кардиналност везе:

1.	1 : 1	_____	ВЛАСНИК – БРОЈ ТЕЛЕФОНА
2.	1 : M	_____	НАСТАВНИК – ПРЕДМЕТ
3.	M : M	_____	ОСОБА – ПАСОШ
		_____	КУПАЦ – МОДЕЛ АУТОМОБИЛА
		_____	УТАКМИЦА – ГРАД ДОМАЋИН

PRIMARNI I SPOLJAŠNJI KLJUČ

Primarni ključ u bazama podataka je jedinstveni identifikator koji se koristi za prepoznavanje svakog reda u tabeli. To je obično jedan atribut u tabeli koji ima jedinstvenu vrednost za svaki red (Osoba – JMBG, Automobil – Broj šasije, Korisnik – Broj računa ...). Primarni ključ omogućuje bazi podataka da brzo pronađe i ažurira specifični red u tabeli.

Primarni ključ je važan jer omogućuje jednostavno pretraživanje i izmenu podataka u bazi. Bez primarnog ključa, baza podataka može imati probleme s traženjem i manipulacijom podacima. Takođe je važno napomenuti da primarni ključ ne može biti prazan ili NULL vrednost.

Primarni ključ se može koristiti za povezivanje podataka između različitih tabela u bazi podataka. Na primer, primarni ključ u tabeli "Osoba" može se koristiti za povezivanje s primarnim ključem u tabeli "Transakcija", što omogućuje bazi podataka da brzo pretražuje i prikazuje sve transakcije koje su povezane s određenom osobom.

Spoljašnji ključ u bazama podataka je atribut ili kombinacija atributa u jednoj tabeli koji se koristi za povezivanje podataka sa primarnim ključem u drugoj tabeli. Spoljašnji ključevi se koriste za stvaranje veza između različitih tabela u bazi podataka, omogućavajući bazi podataka da prikaže podatke koji su međusobno povezani.

Primarni ključ u jednoj tabeli se obično koristi kao spoljašnji ključ u drugoj tabeli. Spoljašnji ključevi povezuju redove dve tabele na temelju zajedničkih vrednosti atributa. Na primer, ako je primarni ključ u tabeli "Osoba" ID broj osobe, spoljašnji ključ u tabeli "Transakcija" mogao bi biti ID broj osobe kako bi se povezali s transakcijama koje je obavila određena osoba.

Vrednost u polju spoljašnjeg ključa mora biti ili NULL ili jednaka nekoj od vrednosti iz kolone na koju spoljašnji ključ referencira. Više redova u tabeli može sadržati istu vrednost u polju spoljašnjeg ključa i time pokazivati na isti red u referenciranoj tabeli (Npr. više učenika sa istim ID brojem smera, više gradova sa istim ID brojem države...). Kolona spoljašnjeg ključa mora sadržati isti tip podataka kao kolona na koju spoljašnji ključ referencira.

Pitanja iz priručnika za maturski ispit

209. Заокружите тачан исказ:

1. Примарни кључ је атрибут који мора бити целобројног типа.
2. Примарни кључ је атрибут који указује на зависност од неке друге табеле.
3. Примарни кључ је атрибут који јединствено идетификује врсте у табели.
4. Ако табела садржи вишевредносни атрибут, њему се додељује функција примарног кључа.

244 Обележити ентитете код којих је извршен адекватан избор јединственог идентификатора. Заокружи бројеве испред тражених одговора:

1. јединствени матични број грађанина (ЈМБГ) за ентитет ОСОБА
2. датум рођења за ентитет ОСОБА
3. ИСБН број за ентитет КЊИГА
4. регистарска ознака за АУТОМОБИЛ
5. дестинација за ентитет АРАНЖМАН
6. режисер за ентитет ФИЛМ

232 Дати су искази који се односе на спољашњи кључ табеле (*foreign key constraint*). Заокружити бројеве испред **тачних** исказа:

1. Вредност у пољу спољашњег кључа не сме бити NULL
2. Вредност спољашњег кључа мора бити јединствена (unique) у колони над којом је постављено ограничење спољашњег кључа
3. Вредност у пољу спољашњег кључа мора бити или NULL или једнака некој од вредности из колоне на коју спољашњи кључ референцира
4. Више редова у табели може садржати исту вредност у пољу спољашњег кључа и тиме показивати на исти ред у референцираној табели
5. Колона спољашњег кључа не мора садржати исти тип података као колона на коју спољашњи кључ референцира

NORMALIZACIJA

Normalne forme u bazama podataka su pravila koja se primenjuju na dizajniranje tabela u cilju minimiziranja redundancije podataka i sprečavanja anomalija baza podataka. Postoje četiri normalne forme: Prva normalna forma (1NF), Druga normalna forma (2NF), Treća normalna forma (3NF) i Bose-Coddova normalna forma (BCNF).

Prva normalna forma (1NF)

Ova normalna forma zahteva da svaka kolona u tabeli ima samo jednu vrednost i da svaki red u tabeli ima jedinstven identifikator (primarni ključ). Tabela koja ne ispunjava ovaj uslov smatra se neprojektovanom ili neredundantnom.

Druga normalna forma (2NF)

Ova normalna forma zahteva da svaka kolona u tabeli zavisi samo od primarnog ključa, a ne od nekog drugog atributa. Tabela koja ispunjava prvu normalnu formu i u kojoj se ne javljaju delimične zavisnosti smatra se u drugoj normalnoj formi.

Treća normalna forma (3NF)

Ova normalna forma zahteva da svaka kolona u tabeli zavisi samo od primarnog ključa i ne zavisi od drugih neprimarnih ključeva. Tabela koja ispunjava drugu normalnu formu i u kojoj se ne javljaju tranzitivne zavisnosti smatra se u trećoj normalnoj formi.

Bose-Coddova normalna forma (BCNF)

Ova normalna forma zahteva da svaka funkcionalna zavisnost u tabeli ima ključ kao kandidata, što znači da svaki neprimarni atribut funkcionalno zavisi samo od primarnog ključa. Tabela koja ispunjava treću normalnu formu i u kojoj se ne javljaju zavisnosti zasnovane na kandidatima smatra se u Bose-Coddovoj normalnoj formi.

Primena ovih normalnih formi omogućava da se naprave baze podataka sa minimalnom redundancijom podataka, smanjuju se šanse za greške u podacima i olakšava se održavanje i modifikacija baze podataka.

Pitanja iz priručnika za maturski ispit

247. Допунити реченицу наводећи назив нормалне форме:

Уколико ни један атрибут релације није вишевредносни, нити композитни, тј. не може се раставити, кажемо да је релација у _____ нормалној форми.

---

248. Допунити реченицу наводећи назив нормалне форме:

Уколико сви атрибути релације који нису део кључа зависе од сваког атрибута који је део кључа кажемо да је релација у _____ нормалној форми.

---

249. Допунити реченицу наводећи назив нормалне форме:

Уколико сви некључни (споредни) атрибути релације не зависе од неког другог некључног атрибута, тј. ако не постоји транзитивна зависност било ког споредног атрибута од било ког кључа те релације, кажемо да је релација у _____ нормалној форми.

SQL KOMANDE

Kategorije SQL komandi

Kategorije SQL komandi su :

DDL (Data Definition Language) – CREATE, ALTER, DROP

DML (Data Manipulation Language) – SELECT, INSERT, UPDATE, DELETE

DCL (Data Control Language) – GRANT, REVOKE

TCL (Transaction Control Language) – COMMIT, ROLLBACK, SAVEPOINT

Naredbe kreiranja, brisanja i izmene strukture baze podataka

Naredbe koje služe za kreiranje, brisanje i izmenu strukture relacione baze podataka i objekata koji čine relacionu bazu su sledeće:

CREATE - koristi se za stvaranje novih objekata u bazi podataka.

DROP - koristi se za brisanje postojećih objekata iz baze podataka.

ALTER - koristi se za izmenu postojećih objekata u bazi podataka.

Primeri naredbi:

CREATE TABLE - koristi se za stvaranje nove tabele u bazi podataka.

DROP TABLE - koristi se za brisanje postojeće tabele iz baze podataka.

ALTER TABLE - koristi se za izmenu strukture postojeće tabele, kao što su dodavanje novih kolona, brisanje kolona ili izmena tipa podataka kolona.

Komande ažuriranja podataka

Komande ažuriranja u bazama podataka su SQL naredbe koje se koriste za izmenu ili brisanje postojećih podataka u tabelama baze podataka.

Neki od najčešćih SQL izraza koji se koriste za ažuriranje podataka su:

UPDATE - koristi se za izmenu postojećih podataka u tabeli.

DELETE - koristi se za brisanje postojećih podataka iz tabele.

INSERT - koristi se za dodavanje novih podataka u tabelu.

Ako se odnosi na SQL bazu podataka, nekoliko osnovnih komandi koje se mogu koristiti za ažuriranje postojećih podataka su:

UPDATE - Koristi se za ažuriranje postojećih podataka u tabeli.

Primer:

```
UPDATE naziv_tabele SET red='nova vrednost' WHERE uslov;
```

Napomena: uslov ograničava koje redove treba ažurirati u tabeli.

MERGE – Koristi se za kombinovanje podataka iz dve tabele. Ova operacija se koristi kada želite ažurirati ili umetnuti podatke u jednu tabelu na osnovu podataka iz druge tabele.

Uopšteno, MERGE komanda izvršava tri osnovne operacije :

Pronalazi odgovarajuće zapise u ciljnoj tabeli za svaki zapis iz izvorne tabele na osnovu definisanog uslova (obično primarni ključ).

Ako postoji odgovarajući zapis u ciljnoj tabeli, ažurira taj zapis prema vrednostima iz izvorne tabele.

Ako ne postoji odgovarajući zapis u ciljnoj tabeli, umetne novi zapis u ciljnu tabelu na osnovu vrednosti iz izvorne tabele.

Pitanja iz priručnika za maturski ispit

256. На левој страни су наведене категорије SQL команди, а са десне су набројане команде. На линију испред команде уписати број под којим је наведена категорија којој команда припада:

1. DDL – Data Definition Language
2. DML – Data Manipulation Language
3. DCL – Data Control Language
4. TCL – Transaction Control Language

_____ GRANT
_____ UPDATE
_____ COMMIT
_____ DROP
_____ DELETE
_____ ALTER

228 Обележити команде које се сматрају командама ажурирања података у бази података. Заокружити бројеве испред тражених одговора:

1. Организовање податка
2. Додавање нових података
3. Брисање старих података
4. Враћање оштећених података у коректно стање
5. Измена постојећих података
6. Додела права приступа подацима
7. Измена структуре постојећих табела у бази

230 Заокружити бројеве испред наредби које служе за креирање, брисање и измену структуре релационе базе и објеката који чине релациону базу:

1. ALTER TABLE
2. INSERT
3. CREATE TABLE
4. DROP TABLE
5. UPDATE
6. DELETE TABLE
7. ADD COLUMN
8. ADD CONSTRAINT

215. Дата је табела *PROJEKAT* над којом се извршава упит:

```
ALTER TABLE PROJEKAT ADD RokKraj date
```

Одредити шта ће се десити након извршења упита. Заокружити број испред траженог одговора.

1. у табелу PROJEKAT додаје се ограничење RokKraj
2. у табелу PROJEKAT додаје се колона RokKraj
3. у табели PROJEKAT биће преименована колона
4. у базу података додаје се табела PROJEKAT са само једном колоном
5. у табели PROJEKAT промениће се тип података у колони RokKraj

239 Заокружити бројеве испред команди које се могу користити за ажурирање постојећих података у бази:

1. DELETE
2. MERGE
3. SELECT
4. UPDATE

235 Дата је табела ***RADNIK***, табела ***ODELJENJE*** и упит:

IDBR	IME	PREZIME	PLATA	SIFRAOD
5900	Slobodan	Golubović	900	10
5932	Mitar	Gavrilović	600	
5953	Persida	Kosanović	1100	20

SIFRAOD	IMEOD	MESTO
10	Marketing	Vračar
20	Direkcija	Grocka
30	Nabavka	Barajevo

Креирана је ускладиштена процедура са параметром `@br int = NULL`

Позивом процедуре извршава се упит:

```
UPDATE Radnik SET Radnik.SifraOD = 30
WHERE Radnik.SifraOD=@br or @br IS NULL
```

Одредити који од понуђених исказа су тачни. Заокружити бројеве испред тражених одговора:

1. Сви радници који раде у одељењу са шифром једнакој вредности која је пренета кроз параметар @br, биће прераспоређени у одељење чија је шифра 30
2. Упит распоређује све нераспоређене раднике у одељење са шифром 30
3. Уколико се параметру @br не пренесе вредност, радници који су до тог момента били нераспоређени, биће распоређени у одељење са шифром 30
4. Уколико се параметру @br не пренесе вредност, СВИ радници ће бити прераспоређени у одељење чији је brod 30

AGREGATNE FUNKCIJE

Agregatne funkcije u SQL-u su funkcije koje se koriste za izračunavanje agregiranih (grupisanih) podataka iz baze podataka. Ove funkcije su vrlo korisne kada želimo da izračunamo statističke podatke, kao što su broj redova, srednja vrednost, maksimalna i minimalna vrednost, zbir i slično.

Najčešće korišćene agregatne funkcije u SQL-u su:

COUNT - broji broj redova u određenoj tabeli ili u grupi

SUM - izračunava zbir vrednosti u određenoj koloni ili grupi

AVG - izračunava prosečnu vrednost u određenoj koloni ili grupi

MAX - pronalazi maksimalnu vrednost u određenoj koloni ili grupi

MIN - pronalazi minimalnu vrednost u određenoj koloni ili grupi

Sintaksa upotrebe ovih funkcija je vrlo jednostavna, a obično se koriste u kombinaciji sa klauzulom GROUP BY koja grupiše podatke po određenom kriterijumu.

Na primer, ako želimo da izračunamo broj knjiga koje se nalaze u biblioteci po kategoriji, možemo koristiti sledeći SQL upit:

```
SELECT kategorija, COUNT(*) as broj_knjiga
FROM Knjiga
GROUP BY kategorija;
```

Ovaj upit će nam vratiti broj knjiga po kategoriji iz tabele "Knjiga" i grupisati ih po kategorijama.

SQL KLAUZULE

SQL klauzule su delovi SQL naredbi koje se koriste za filtriranje, sortiranje ili grupisanje podataka u bazi podataka. One se koriste u sklopu SQL naredbi kao što su SELECT, INSERT, UPDATE ili DELETE kako bi se specificirale dodatne operacije koje se trebaju izvršiti nad podacima u bazi podataka.

Neke od najčešće korišćenih SQL klauzula su:

JOIN - koristi se za spajanje podataka iz dve ili više tabela

WHERE - koristi se za filtriranje redova podataka koji ispunjavaju određene kriterijume

GROUP BY - koristi se za grupisanje podataka po određenim kriterijumima

HAVING - koristi se za filtriranje grupisanih podataka koji ispunjavaju određene kriterijume

ORDER BY - koristi se za sortiranje podataka po određenom kriterijumu

Ove klauzule se mogu kombinovati na različite načine kako bi se dobili željeni rezultati iz baze podataka.

Join-ovi

Join-ovi u SQL-u su mehanizmi za povezivanje podataka iz dve ili više tabela u bazi podataka. Uobičajeno je da imamo podatke koji su podeljeni u više tabela, a da bismo dobili potpunu sliku, često je potrebno izvršiti spajanje (join) tih tabela.

Postoje različite vrste join-ova u SQL-u, a neke od najčešće korišćenih su:

INNER JOIN : Vraća samo one redove koji se podudaraju u oba niza, tj. samo one redove gde postoji podudaranje između ključa u obe tabele.

LEFT JOIN : Vraća sve redove iz leve tabele i sve podudarajuće redove iz desne tabele. Ako u desnoj tabeli ne postoji podudaranje, vrednosti iz desne tabele će biti NULL.

RIGHT JOIN : Vraća sve redove iz desne tabele i sve podudarajuće redove iz leve tabele. Ako u levoj tabeli ne postoji podudaranje, vrednosti iz leve tabele će biti NULL.

FULL OUTER JOIN : Vraća sve redove iz obe tabele. Ako u bilo kojoj tabeli ne postoji podudaranje, vrednosti će biti NULL.

Sintaksa upotrebe join-ova u SQL-u je obično sledeća:

```
SELECT kolone
FROM tabela1
JOIN tabela2 ON uslov
```

U ovom primeru, "tabela1" i "tabela2" su tabele koje želimo da spojimo, a "uslov" je uslov spajanja koji obično sadrži ključeve po kojima se vrši spajanje. Takođe, možemo koristiti klauzule WHERE i GROUP BY kako bismo dodatno filtrirali i grupisali podatke koje dobijamo spajanjem tabela.

Na primer, ako želimo da spojimo tabele "Knjiga" i "Autor" po autor_id koloni i prikažemo nazive knjiga, autora i godinu izdanja, možemo koristiti sledeći SQL upit:

```
SELECT Knjiga.naziv, Autor.ime, Knjiga.godina_izdavanja
FROM Knjiga
JOIN Autor ON Knjiga.autor_id = Autor.autor_id;
```

Ovaj upit će spojiti tabele "Knjiga" i "Autor" po autor_id koloni, prikazati nazive knjiga, imena autora i godine izdanja za sve podudarajuće redove.

Where

WHERE klauzula u SQL-u se koristi za filtriranje redova u tabeli na osnovu određenog uslova.

Na primer, ako želimo da izlistamo samo one redove u tabeli "Učenik" koji imaju ocenu veću od 3, upotrebili bismo WHERE klauzulu. SQL upit bi izgledao ovako:

```
SELECT * FROM Učenik WHERE ocena > 3;
```

Ovaj upit bi vratio samo one redove iz tabele "Učenik" gde je vrednost u koloni "ocena" veća od 8.

U WHERE klauzulu možemo koristiti različite operatora za poređenje, kao što su "=" (jednako), "<" (manje od), ">" (veće od), "<=" (manje ili jednako), ">=" (veće ili jednako), "<>" (različito od), kao i logičke operatore kao što su "AND", "OR" i "NOT".

Group by

Group by u SQL-u je funkcija koja omogućava grupisanje redova u tabeli prema nekoj koloni ili kolonama. Kada koristimo funkciju Group by, SQL nam vraća rezultat u kome su redovi grupisani prema vrednostima u kolonama koje smo odabrali.

Na primer, ako imamo tabelu koja sadrži informacije o prodaji proizvoda, sa kolonama "proizvod", "količina" i "cena", možemo koristiti funkciju Group by da bismo grupisali prodaju po proizvodu i dobili ukupnu količinu i ukupnu cenu za svaki proizvod.

Sintaksa za funkciju Group by je sledeća:

```
SELECT kolona1, kolona2, ..., funkcija(kolona)
FROM ime_tabele
GROUP BY kolona1, kolona2, ...
```

Ovde, "kolona1, kolona2, ..." predstavljaju kolone po kojima želimo da grupišemo podatke, a "funkcija(kolona)" predstavlja funkciju koju želimo da primenimo na grupisane podatke. Na primer, možemo koristiti funkciju SUM da bismo dobili ukupnu količinu ili ukupnu cenu za svaku grupu.

Važno je napomenuti da kada koristimo funkciju Group by, SQL neće vratiti individualne redove u tabeli, već samo grupisane vrednosti. Takođe, ne možemo koristiti kolone koje nismo naveli u funkciji Group by ili u nekoj od funkcija koje primenjujemo na podatke.

Na kraju, funkcija Group by je veoma korisna za analizu podataka u tabelama i može nam pomoći da lako dobijemo pregled grupisanih podataka po određenim kategorijama.

Having

HAVING klauzula se koristi u SQL upitu za filtriranje grupisanih podataka koji su prethodno izdvojeni pomoću GROUP BY klauzule. U suštini, HAVING funkcioniše slično kao WHERE klauzula, ali umesto na pojedinačne redove, primenjuje se na grupe podataka. HAVING se koristi za postavljanje uslova na grupe koje su formirane pomoću GROUP BY klauzule.

Na primer, ako imamo tabelu prodaje sa poljima prodavac, kupac i iznos, a želimo da izdvojimo samo one prodavce čija je ukupna prodaja veća od 10.000 dinara, možemo koristiti sledeći SQL upit:

```
SELECT prodavac, SUM(iznos) as ukupna_prodaja
FROM Prodaja
GROUP BY prodavac
HAVING SUM(iznos) > 10000;
```

U ovom primeru, GROUP BY klauzula grupiše prodavce prema njihovim imenima, a SUM funkcija izračunava ukupnu prodaju za svakog prodavca. Zatim se HAVING klauzulom primenjuje filter da bismo izdvojili samo one prodavce čija je ukupna prodaja veća od 10.000 dinara.

Ukratko, HAVING klauzula se koristi za filtriranje grupisanih podataka u SQL upitu, na osnovu agregiranih vrednosti koje su definisane pomoću funkcija kao što su SUM, COUNT, AVG, MAX i MIN.

Order by

Order by u SQL-u je funkcija koja se koristi da bi se sortirali rezultati upita po određenoj koloni. Kada koristimo funkciju Order by, SQL sortira redove u tabeli po vrednostima u odabranoj koloni.

Na primer, ako imamo tabelu sa informacijama o zaposlenima, sa kolonama "ime", "prezime" i "plata", možemo koristiti funkciju Order by da bismo sortirali zaposlene po plati od najmanje ka najvećoj ili od najveće ka najmanjoj.

Sintaksa za funkciju Order by je sledeća:

```
SELECT kolona1, kolona2, ...
FROM ime_tabele
ORDER BY kolona1 [ASC|DESC], kolona2 [ASC|DESC], ...
```

Ovde, "kolona1, kolona2, ..." predstavljaju kolone po kojima želimo da sortiramo podatke. Kada koristimo više kolona, SQL prvo sortira po prvoj koloni, a zatim po drugoj i tako dalje. "ASC" znači sortiranje od najmanje ka najvećoj, a "DESC" od najveće ka najmanjoj. Podrazumevana vrednost je "ASC".

Važno je napomenuti da kada koristimo funkciju Order by, SQL menja redosled redova u tabeli, ali ne menja samu tabelu. Ovo znači da ne možemo uneti podatke u tabelu koristeći funkciju Order by. Takođe, moguće je koristiti funkcije i izraze u poljima upita, ali oni moraju biti navedeni pre korišćenja funkcije Order by.

Funkcija Order by je veoma korisna za analizu podataka u tabelama i može nam pomoći da lako sortiramo podatke po određenim kolonama.

Aritmetički operatori u klauzulama

U SQL naredbama se aritmetičke operacije mogu koristiti u sledećim klauzulama:

SELECT klauzula - Aritmetičke operacije se mogu koristiti unutar SELECT klauzule kako bi se izračunale nove vrednosti na osnovu postojećih podataka.

Na primer:

```
SELECT kolona1, kolona2, kolona1 + kolona2 AS zbir
FROM tabela
```

Ova naredba će prikazati vrednosti iz kolona "kolona1" i "kolona2", kao i novu kolonu "zbir", koja će biti izračunata kao zbir vrednosti iz kolona "kolona1" i "kolona2".

WHERE klauzula - Aritmetičke operacije se mogu koristiti unutar WHERE klauzule kako bi se filtrirali podaci na osnovu nekog izračuna.

Na primer:

```
SELECT kolona1, kolona2
FROM tabela
WHERE kolona1 + kolona2 > 10
```

Ova naredba će prikazati sve redove iz tabele u kojima je zbir vrednosti iz kolona "kolona1" i "kolona2" veći od 10.

ORDER BY klauzula - Aritmetičke operacije se mogu koristiti unutar ORDER BY klauzule kako bi se sortirali podaci na osnovu nekog izračuna.

Na primer:

```
SELECT kolona1, kolona2
FROM tabela
ORDER BY kolona1 + kolona2 DESC
```

Ova naredba će prikazati sve redove iz tabele sortirane po zbiru vrednosti iz kolona "kolona1" i "kolona2" u opadajućem redosledu.

Pitanja iz priručnika za maturski ispit

253. Уписати редни број почев од 1 на линију испред резервисане речи тако да одговара редоследу навођења при формирању упита.

За формирање упита за издвајање дела података из табеле која се налази у оквиру базе података користе се клаузуле у следећем редоследу:

_____ GROUP BY

_____ WHERE

_____ SELECT

_____ ORDER BY

_____ FROM

222. Извршава се упит:

```
SELECT prezime, ime, email
FROM ucenik
ORDER BY prezime
WHERE prosek>=4.50
```

Наредба се неће извршити. Заокружити број испред разлога услед кога се наредба неће изврити:

1. Наредба се неће извршити једино ако нема ни једног одличног ученика.
2. Услов треба написати у HAVING клаузули
3. Потребно је назначити редослед сортирања (asc, desc).
4. Потребно је променити редослед клаузула.

218. Дата је табела ***RADNIK***, табела ***ODELJENJE*** и упит:

IDBR	IME	PREZIME	PLATA	BROD
5900	Slobodan	Golubović	900	10
5932	Mitar	Gavrilović	600	
6234	Marko	Pavlović	1300	30
6789	Janko	Nikolić	800	10

BROD	IMEOD	MESTO
50	Skladišta	Zemun
30	Marketing	Vračar
10	Plasman	Surčin
20	Direkcija	Grocka

```
SELECT Odeljenje.Imeod, Radnik.Ime+' '+ Radnik.Prezme as PunoIme
FROM Odeljenje LEFT JOIN Radnik ON Radnik.Brod = Odeljenje.Brod
```

Одредити шта се види као резултат датог упита. Заокружити број испред траженог одговора:

1. Називи свих одељења – и оних у којима има радника и оних где нико није распоређен - са бројем радника у сваком одељењу
2. За сваког распоређеног радника приказује се по један ред са називом одељења и пуним именом радника, док се за раднике који нису распоређени приказује само пуно име радника
3. Приказује се по један ред за сваког распоређеног радника са називом одељења и пуним именом радника. За раднике који нису распоређени, као и за одељења у која нико није распоређен, не формирају се редови у резултујућој табели
4. За свако одељење се приказује онолико редова колико радника ради у том одељењу, док се за одељења у којима нико не ради приказује по један ред са називом одељења

219. Дата је табела ***RADNIK***, табела ***ODELJENJE*** и упит:

IDBR	IME	PREZIME	PLATA	BROD
5900	Slobodan	Golubović	900	10
5932	Mitar	Gavrilović	600	10
5953	Persida	Kosanović	1100	20
6234	Marko	Pavlović	1300	30
6789	Janko	Nikolić	800	10

BROD	IMEOD	MESTO
50	Skladišta	Zemun
30	Marketing	Vračar
10	Plasman	Surčin
20	Direkcija	Grocka
40	Nabavka	Barajevo

```
SELECT Odeljenje.Brod, Odeljenje.Imeod, COUNT(*)
FROM Radnik INNER JOIN Odeljenje ON Radnik.Brod = Odeljenje.Brod
GROUP BY Odeljenje.Brod, Odeljenje.Imeod
```

Одредити приказ који је резултат датог упита. Заокружити број испред траженог одговора:

1. Бројева и назива свих одељења
2. Бројева и назива свих одељења са бројем радника у њима
3. Бројева и назива одељења у којима има радника са бројем радника у њима
4. Бројева и назива одељења у којима нема радника

241 Креиране су и попуњене подацима табеле ***Korisnik*** и ***Prijatelji.*** Њихова структура и садржај приказани су на слици:

Korisnik

ID	IME	POL
1	Ana	NULL
2	Steva	m
3	Marta	z
4	Petra	z

Prijatelji

Korisnik1	Korisnik2
1	2
1	3
2	3

Извршавањем упита добија се табела са подацима.

```
select k.ime, COUNT(*) as [broj prijatelja]
from Korisnik as k
left join Prijatelji as p on p.korisnik1=k.id or p.korisnik2=k.id
where k.pol='z'
group by k.id, k.ime
```

Заокруживањем бројева испред понуђених одговора, обележити који од наведених података ће бити приказани у појединим редовима резултујуће табеле:

1. Ana, 1
2. Ana, 2
3. Steva, 1
4. Steva, 2
5. Marta, 1
6. Marta, 2
7. Petra, 0
8. Petra, 1

254. Дата је табела ***RADNIK***, табела ***ODELJENJE*** и упит:

IDBR	IME	PREZIME	PLATA	BROD
5900	Slobodan	Golubović	900	10
5932	Mitar	Gavrilović	600	
5953	Persida	Kosanović	1100	20
6234	Marko	Pavlović	1300	30
6789	Janko	Nikolić	800	10

BROD	IMEOD	MESTO
50	Skladišta	Zemun
30	Marketing	Vračar
10	Plasman	Surčin
20	Direkcija	Grocka
40	Nabavka	Barajevo

Повезати упите и њихова значења уписом броја упита на одговарајућу линију:

1.
```
SELECT odeljenje.imeod,
radnik.prezime
FROM odeljenje INNER JOIN
radnik
ON radnik.brod = odeljenje.brod
```
Приказује све раднике (и који јесу и који нису распоређени у одељења) и само она одељења у којима има радника

2.
```
SELECT odeljenje.imeod,
radnik.prezime
FROM odeljenje LEFT JOIN radnik
ON radnik.brod = odeljenje.brod
```
Приказује само одељења у којима има радника и само раднике распоређене у одељењима

3.
```
SELECT odeljenje.imeod,
radnik.prezime
FROM odeljenje RIGHT JOIN
radnik
ON radnik.brod = odeljenje.brod
```
Приказује сва одељења (и она у којима има и она у којима нема радника) и само оне раднике који су распоређени у одељења

255. Дата је табела ***RADNIK***, табела ***ODELJENJE*** и упит:

IDBR	IME	PREZIME	PLATA	BROD
5900	Slobodan	Golubović	900	10
5932	Mitar	Gavrilović	600	
5953	Persida	Kosanović	1100	20
6234	Marko	Pavlović	1300	30
6789	Janko	Nikolić	800	10

BROD	IMEOD	MESTO
50	Skladišta	Zemun
30	Marketing	Vračar
10	Plasman	Surčin
20	Direkcija	Grocka
40	Nabavka	Barajevo

Повезати упите и њихова значења уписом броја датог испред описа значења упита на одговарајућу линију:

```
SELECT odeljenje.imeod,
radnik.prezime
FROM odeljenje LEFT JOIN radnik
ON radnik.brod = odeljenje.brod
WHERE radnik.brod IS NULL
```

_____

```
SELECT odeljenje.imeod,
radnik.prezime
FROM odeljenje FULL JOIN radnik
ON radnik.brod = odeljenje.brod
```

_____

```
SELECT odeljenje.imeod,
radnik.prezime
FROM odeljenje RIGHT JOIN
radnik
ON radnik.brod = odeljenje.brod
WHERE odeljenje.brod IS NULL
```

_____

1. Приказује само раднике који нису распоређени у одељења
2. Приказује све раднике (и који јесу и који нису распоређени у одељења) и само она одељења у којима има радника
3. Приказује сва одељења - и она у којима има и оне у којима нема радника и све раднике – и оне који су распоређени у одељења, као и оне који нису распоређени
4. Приказује само одељења у којима нема радника

213. Заокруживањем редног броја обележити клаузулу коју је потребно користи уколико листа иза резервисане речи SELECT садржи агрегатну функцију и једну или више колона које нису део агрегатне функције:

1. HAVING клаузулу
2. GROUP BY клаузулу
3. JOIN клаузулу
4. ORDER BY клаузулу

217. Дата је табела ***RADNIK***, табела ***ODELJENJE*** и упит:

IDBR	IME	PREZIME	PLATA	BROD
5900	Slobodan	Golubović	900	10
5932	Mitar	Gavrilović	600	10
5953	Persida	Kosanović	1100	20
6234	Marko	Pavlović	1300	30
6789	Janko	Nikolić	800	10

BROD	IMEOD	MESTO
50	Skladišta	Zemun
30	Marketing	Vračar
10	Plasman	Surčin
20	Direkcija	Grocka
40	Nabavka	Barajevo

```
SELECT Imeod, AVG(Plata) AS ProsekPlata FROM Radnik, Odeljenje
WHERE Odeljenje.Brod=Radnik.Brod GROUP BY Imeod HAVING
AVG(Plata)>1000
```

Одредити резултат извршавања датог упита. Заокружити број испред траженог одговора:

1. Упит се не извршава зато што груписање мора да се изврши не само по називу одељења, него и по шифри одељења (BrOd)
2. Групишу по одељењима радници са платом већом од просечне плате
3. Приказују називи одељења и висина просечне плате у њима само за одељења у којима је просечна плата већа од 1000
4. Приказују називи одељења и висина просечне плате у њима, при чему се код одређивања просека узимају у обзир само плате веће од 1000

220. Извршава се следећа SELECT наредба:

```
SELECT MIN(Datum_Zaposlenja), Odsek_Id
FROM Zaposleni
GROUP BY Odsek_Id
```

Заокруживањем броја испред одговарајућег исказа, одредити које ће вредности бити приказане:

1. Најранији датум запослења за сваки одсек предузећа.
2. Најранији датум запослења у целом предузећу.
3. Датум запослења последњег запосленог радника у целом предузећу.
4. Датум запослења последњег запосленог радника за сваки одсек.
5. Датум запослења најстаријег запосленог радника у сваком одсеку предузећа.

221. Потребно је креирати извештај који приказује имена свих производа чија је цена већа од просечне цене свих производа.

Заокружити број испред упита који одговара постављеном задатку:

1. ```
   SELECT naziv
   FROM proizvod
   WHERE cena > (SELECT AVG(cena) FROM proizvod)
   ```
2. ```
   SELECT naziv
   FROM proizvod
   WHERE cena > AVG(cena)
   ```
3. ```
   SELECT naziv
   FROM proizvod
   GROUP BY naziv
   HAVING cena > AVG(cena)
   ```
4. ```
   SELECT naziv
   FROM (SELECT AVG(cena) FROM proizvod)
   WHERE cena > AVG(cena)
   ```

223. Табела ***ARTIKLI*** садржи следеће колоне: *artikl_id, naziv, kategorija, cena, kolicina*.

Потребно је да се прикаже категорија и минимална цена артикла у свакој категорији. При томе се тражи приказ само оних категорија где је најмања цена производа већа од задате граничне вредности која се преноси упиту кроз параметар *@granica*

Изабрати упит који даје тражени извештај:

1. ```
   SELECT kategorija, MIN(cena)
   FROM artikli
   WHERE MIN(cena)>@granica
   GROUP BY cena
   ```
2. ```
   SELECT kategorija, MIN(cena)
   FROM artikli
   GROUP BY kategorija
   HAVING MIN(cena)>@granica
   ```
3. ```
   SELECT kategorija, MIN(cena)
   FROM artikli
   GROUP BY MIN(cena), kategorija
   HAVING MIN(cena)>@granica
   ```
4. ```
   SELECT kategorija, MIN(cena)
   FROM artikli
   WHERE MIN(cena)>@granica
   GROUP BY kategorija
   ```

224. Табела ***RADIONICA*** садржи следеће колоне: ***radionica_id, naziv, zanat, lokacija_id***.

Потребно је да се прикаже колико на свакој локацији има различитих заната. Заокружити број испред упита који даје тражени извештај:

1. ```
   SELECT DISTINCT location_id, COUNT(zanat)
   FROM radionica
   GROUP BY lokacija_id
   ```
2. ```
   SELECT location_id, COUNT(zanat)
   FROM radionica
   GROUP BY lokacija_id
   ```
3. ```
   SELECT location_id, COUNT(DISTINCT zanat)
   FROM radionica
   GROUP BY lokacija_id
   ```
4. ```
   SELECT location_id, COUNT(DISTINCT zanat)
   FROM radionica
   GROUP BY zanat
   ```

236 Табела **Zaposleni** садржи поља: Zaposleni_Id, Ime, Prezime, Plata, Odsek_Id.

Дат је упит:

```
SELECT Zaposleni_Id, Ime, Prezime
FROM Zaposleni
WHERE Plata=(SELECT MAX(Plata) FROM Zaposleni GROUP BY Odsek_Id)
```

Дати су искази који описују ефекат извршења упита. Заокружити бројеве испред ТАЧНИХ исказа:

1. Упит се не извршава зато што у подупиту није дозвољено коришћење групних функција
2. Упит се извршава без грешке и из сваког одељења бира и приказује податке о раднику који има највећу плату у том одељењу.
3. Упит се не извршава јер подупит враћа више од једне врсте, а коришћен је оператор за поређење са једном вредношћу.
4. Уколико би се изоставила GROUP BY клаузула, упит би се извршавао без грешке и приказао би једног или више радника са платом једнакој највећој плати (без обзира на одељење).
5. Како подупит садржи груписање, да би се цео упит извршио без грешке, потребно је услов са подупитом написати у HAVING уместо у WHERE клаузули

229 Заокружити бројеве испред тражених одговора.

За упите са специфицираним редоследом приказа врста у резултујућој табели користи се клаузула ORDER BY после које се наводи назив колоне:

1. и службена реч ASC за растући редослед
2. и службена реч DESC за опадајући редослед
3. и службена реч ASC за опадајући редослед
4. и службена реч DESC за растући редослед
5. службена реч се може изоставити при чему се добија растући поредак
6. службена реч се може изоставити при чему се добија опадајући поредак

237 Заокружити бројеве под којима су наведене клаузуле SQL наредбе у којима се могу користити аритметичке операције:

1. SELECT
2. FROM
3. WHERE
4. ORDER BY

LOGIČKE OPERACIJE

NOT – Negacija, vraća DA ako je uslov NE. Logički operator najvišeg prioriteta.

AND – Logičko I, vraća DA ako su svi uslovi koji se ispituju tačni. Logički operator srednjeg prioriteta.

OR – Logičko ILI, vraća DA ako je bar jedan od uslova koji se ispituju tačan. Logički operator najnižeg prioriteta.

Pitanja iz priručnika za maturski ispit

250. Написати на цртама испред логичких операција редне бројеве њихових приоритета:

1. највиши приоритет _____ OR
2. средњи приоритет _____ NOT
3. најнижи приоритет _____ AND

SQL OPERATORI

Operator IN

Operator IN u SQL-u se koristi kako bi se proverilo da li se vrednost kolone nalazi u listi vrednosti.

Koristi se u WHERE klauzuli, a sintaksa je obično:

```
SELECT ime_kolone(kolona)
FROM ime_tabele
WHERE ime_kolone IN (vrednost1, vrednost2, ...);
```

Na primer, ako imamo tabelu "Korisnik" sa kolonama "Ime", "Prezime" i "Grad", i želimo da pronađemo sve korisnike iz Beograda ili Novog Sada, možemo koristiti operator IN ovako:

```
SELECT Ime, Prezime
FROM Korisnik
WHERE Grad IN ('Beograd', 'Novi Sad');
```

Ovaj upit će izvući sve korisnike čiji je grad "Beograd" ili "Novi Sad".

Možemo takođe koristiti operator IN sa podupitima, što znači da možemo uporediti vrednosti u jednoj tabeli sa vrednostima u drugoj tabeli.
Na primer, ako imamo tabelu "Korisnik" sa kolonama "Ime", "Prezime" i "Grad", i želimo da pronađemo sve korisnike koji su kupili proizvode iz liste proizvoda koju smo definisali u drugoj tabeli "Proizvod", možemo koristiti operator IN ovako:

```
SELECT Ime, Prezime
FROM Korisnik
WHERE KorisnikID IN (SELECT KorisnikID FROM Narudzbina WHERE ProizvodID IN (SELECT ProizvodID FROM Proizvod
WHERE Naziv IN ('Proizvod1', 'Proizvod2', 'Proizvod3')));
```

Ovaj upit će izvući sve korisnike koji su kupili proizvode čiji nazivi se nalaze na listi "Proizvod1", "Proizvod2" ili "Proizvod3".

Operator IN je koristan kada želimo da izvučemo podatke na osnovu liste vrednosti ili kada želimo da uporedimo vrednosti u jednoj tabeli sa vrednostima u drugoj tabeli.

Operator LIKE

Operator LIKE se koristi u SQL-u za pretraživanje vrednosti koje odgovaraju određenom uzorku. Uobičajeno se koristi s nizovima (stringovima), ali se može koristiti i sa drugim tipovima podataka koji podržavaju uzorak koji se pretražuje.

Kada koristite operator LIKE, možete navesti uzorak sastavljen od jednog ili više znakova koji predstavljaju mesta gde se može nalaziti bilo koji drugi znak. Najčešće se koriste dva posebna znaka: "%", koji predstavlja nula ili više znakova, i "_", koji predstavlja tačno jedan znak.

Na primer, ako imate kolonu u bazi podataka koja sadrži imena korisnika, a želite pronaći sve korisnike čija imena počinju slovom "J", možete koristiti sljedeći SQL upit:

```
SELECT * FROM korisnici WHERE ime LIKE 'J%'
```

Ovaj upit će vratiti sve redove iz tabele "korisnici" gde vrednost kolone "ime" počinje slovom "J". Operator "%", koji se koristi iza slova "J", znači da iza njega može doći bilo koji niz znakova.

Ako želite pronaći sve korisnike čija imena završavaju slovom "n", možete koristiti sljedeći upit:

```
SELECT * FROM korisnici WHERE ime LIKE '%n'
```

Ovaj upit će vratiti sve redove iz tabele "korisnici" gde se vrednost kolone "ime" završava slovom "n". Operator "%", koji se koristi pre slova "n", znači da ispred njega može doći bilo koji niz znakova.

Operator BETWEEN

Operator BETWEEN u SQL-u se koristi za filtriranje podataka u određenom rasponu vrednosti. Operator se koristi u WHERE klauzuli SELECT upita, a koristi se u kombinaciji sa operatorom AND.

Sintaksa operatora BETWEEN izgleda ovako:

```
value BETWEEN donja_granica AND gornja_granica
```

Ovde, value predstavlja vrednost koju želimo da uporedimo sa određenim rasponom vrednosti, dok donja_granica i gornja_granica predstavljaju granice raspona.

Na primer, ako želimo da pronađemo sve korisnike koji su stariji od 18 godina i mlađi od 30 godina, koristimo operator BETWEEN na sledeći način:

```
SELECT * FROM Korisnik WHERE godine BETWEEN 18 AND 30;
```

Ovo će vratiti sve redove iz tabele Korisnik u kojima se kolona godine nalazi u rasponu od 18 do 30 godina.

Važno je napomenuti da operator BETWEEN uključuje granice, što znači da će se i vrednosti koje se podudaraju sa granicama uključiti u rezultat.

Operator SOME

Operator SOME u SQL-u se koristi u kombinaciji sa drugim operatorima (npr. =, >, <, >=, <=, itd.) kako bi se vršilo poređenje sa vrednostima koje se vraćaju iz podupita.

Sintaksa je:

```
SELECT ime_kolone(kolona)
FROM ime_tabele
WHERE ime_kolone operator SOME (SELECT ime_kolone FROM ime_tabele WHERE uslov);
```

Operator SOME može se zameniti sa operatorom ANY. Oni su suštinski isti, a operator SOME se koristi u Transact-SQL, dok se operator ANY koristi u drugim implementacijama SQL-a.

Na primer, ako želimo da pronađemo sve zaposlene čija je plata veća od plate bilo kog menadžera, možemo koristiti sledeći SQL upit:

```
SELECT *
FROM Zaposleni
WHERE plata > SOME (SELECT plata FROM Zaposleni WHERE pozicija = 'Menadžer');
```

Ovaj upit će pronaći sve zaposlene čija je plata veća od plate bilo kog menadžera koji rade u istoj tabeli "Zaposleni".

Operator ANY

Operator ANY u SQL-u se koristi kako bi se uporedile vrednosti u podupitu sa vrednostima u glavnom upitu, koristeći neki od operatora poređenja kao što su =, <>, >, <, >=, <=.

Operator ANY omogućava poređenje vrednosti u podupitu sa jednom ili više vrednosti u glavnom upitu. Sintaksa za korišćenje operatora ANY je sledeća:

```
SELECT ime_kolone(kolona)
FROM ime_tabele
WHERE ime_kolone operator ANY (SELECT ime_kolone FROM ime_tabele WHERE uslov);
```

Na primer, ako imamo tabelu "Korisnik" sa kolonama "Ime", "Prezime" i "Starost", i želimo da pronađemo sve korisnike čija je starost jednaka starosti nekog korisnika čiji je ID jednak 3, možemo koristiti operator ANY ovako:

```
SELECT Ime, Prezime
FROM Korisnik
```

```
WHERE Starost = ANY (SELECT Starost FROM Korisnik WHERE KorisnikID = 3);
```

Ovaj upit će izvući sve korisnike čija je starost jednaka starosti korisnika sa ID-jem 3.

Operator ALL

Operator ALL u SQL-u se koristi u kombinaciji sa upitom koji uključuje uslov. On omogućava da se uslov primeni na sve redove koji se vraćaju iz podupita. Konkretnije, operator ALL će uslov primeniti na sve vrednosti u koloni koju poredimo sa drugom vrednošću. Ukoliko se uslov ispuni za sve vrednosti, vraća se TRUE, inače se vraća FALSE.

Evo jednostavnog primera kako bismo to mogli primeniti u praksi. Recimo da imamo tabelu "Proizvod" koja sadrži informacije o proizvodima, kao što su naziv, cena i količina. Sada želimo da pronađemo sve proizvode koji su jeftiniji od 10 dinara.

```
SELECT naziv FROM Proizvod
WHERE cena < ALL (10);
```

Ovaj SQL upit će nam vratiti sve nazive proizvoda čija je cena manja od 10 dinara. Operator ALL nam govori da primenimo uslov "cena manja od 10" na sve vrednosti u koloni "cena", a ne samo na jednu vrednost.

Pitanja iz priručnika za maturski ispit

210. Заокружити број испред траженог одговора.

Одредити оператор који би требало користити у WHERE клаузули SELECT наредбе да би били приказани само они ученици чије презиме почиње словом А:

1. IN
2. LIKE
3. BETWEEN
4. IS LIKE
5. BEGINS WITH

212. Дат је упит:

```
SELECT *
FROM ucenici
WHERE odeljenje=4 OR odeljenje=7 OR odeljenje=10
```

Заокружити оператор који треба користити у датом упиту да би се избегло вишеструко коришћење оператора OR:

1. LIKE
2. BETWEEN
3. AND
4. IN

238 Извршава се следећи упит:

```
SELECT cena
FROM proizvod
WHERE cena IN (101,125,150,350)
AND (cena BETWEEN 125 AND 140 OR cena >150)
```

Одредити које две вредности може вратити ова наредба. Заокружити бројеве испред тражених вредности:

1. 101
2. 150
3. 125
4. 110
5. 350

245 Одредити тачан исказ о оператору ANY који се примењује са подупитом који враћа више вредности:

1. Оператор ANY може да се користи испред кључне речи DISTINCT.
2. Оператор ANY врши поређење са свим вредностима које враћа подупит и враћа TRUE ако све вредности подупита задовољавају услов
3. Оператор ANY врши поређење са свим вредностима које враћа подупит и враћа TRUE ако било која од вредности подупита задовољава услов
4. Оператору ANY може да претходи оператор LIKE или оператор IN.
5. Оператору ANY мора да претходи оператор поређења (=, <>, >, >=, <, <=)
6. Услов =ANY(*скуп вредности*) је еквивалентан услову IN (*скуп вредности*)

251. Дата је табела ***GEOGRAFIJA*** која поред осталих података садржи називе градова и држава (`Naziv nvarchar(50)`). У зависности од услова у ***WHERE*** клаузули, ***SELECT*** упитом се приказују географски појмови из табеле. Са леве стране су дати услови нумерисани бројевима од 1 до 5, а са десне групе градова.

Свакој групи градова придружити по један услов уносом редног броја коим је услов нумерисан на линију испред листе градова:

1. `where Naziv like 'L__ %'` _____ SIJERA LEONE, SVETA LUCIJA
2. `where Naziv like '__ %N%'` _____ LA VALETA, LA KORUNJA
3. `where Naziv like '% L%'` _____ EL RENO, LA KORUNJA
4. `where Naziv like '_L%'` _____ EL SALVADOR, EL RENO
5. `where Naziv like '__ %A'` _____ LAS VEGAS, LOS ANGELES

NULL VREDNOSTI

NULL je vrednost koja se koristi u SQL-u kada ne postoji validna vrednost za neku kolonu u tabeli. Ovo može biti zbog toga što podatak nije poznat, nije primenljiv ili nije dostupan. NULL se može koristiti za sve vrste podataka, uključujući brojeve, tekst, datume, itd.

IS NULL je operator koji se koristi za proveru da li je vrednost kolone NULL. Na primer, ako želimo da pronađemo sve korisnike u tabeli users koji nemaju unetu adresu, možemo koristiti sledeći upit:

```
SELECT * FROM Korisnik WHERE adresa IS NULL;
```

IS NOT NULL je operator koji se koristi za proveru da li vrednost kolone nije NULL. Na primer, ako želimo da pronađemo sve korisnike koji imaju unetu adresu, možemo koristiti sledeći upit:

```
SELECT * FROM Korisnik WHERE adresa IS NOT NULL;
```

Važno je napomenuti da se NULL vrednosti ne mogu uporediti koristeći obične operatore poređenja (npr. =, <, >). Umesto toga, koristimo specijalne funkcije poput IS NULL i IS NOT NULL za proveru da li je vrednost NULL.

Pitanja iz priručnika za maturski ispit

211. Заокружити број испред траженог одговора.

Табела UCENICI поред осталих података, садржи и вредност стипендије. Одредити оператор који треба употребити у WHERE клаузули SELECT наредбе да би били приказани сви ученици код којих није позната и није унета вредност у колону ***stipendija***:

1. =NULL
2. ISNULL
3. ==NULL
4. IS NULL
5. LIKE NULL

RAD SA DATUMIMA

GETDATE()

GETDATE() funkcija u SQL-u se koristi kako bi se dobila trenutna vremenska oznaka u bazi podataka. Sintaksa za korišćenje GETDATE() funkcije je sledeća:

```
SELECT GETDATE();
```

Ovaj upit će vratiti trenutni datum i vreme u bazi podataka. GETDATE() funkcija je često korisna kada želimo da ažuriramo vrednosti datuma i vremena u tabeli ili da uporedimo datume sa trenutnim vremenom.

Na primer, ako želimo da pronađemo sve redove u tabeli "Narudžbine" koji su napravljeni danas, možemo koristiti ovaj upit:

```
SELECT * FROM Narudžbine WHERE DatumNarudžbine = CONVERT(date, GETDATE());
```

Ovaj upit će vratiti sve redove u tabeli "Narudžbine" koji su napravljeni danas, pri čemu se funkcija CONVERT() koristi kako bi se uklonilo vreme iz trenutne vremenske oznake koju vraća GETDATE() funkcija.

DATEDIFF()

DATEDIFF funkcija u SQL-u se koristi kako bi se izračunala razlika između dva datuma u određenoj jedinici vremena, kao što su dani, meseci ili godine. Sintaksa za korišćenje DATEDIFF funkcije je sledeća:

```
DATEDIFF(interval, početni_datum, krajnji_datum)
```

Gde je "interval" jedinica vremena u kojoj se traži razlika, "početni_datum" početni datum, a "krajnji_datum" završni datum.

Na primer, ako želimo da izračunamo broj dana između dva datuma, koristimo DATEDIFF funkciju ovako:

```
SELECT DATEDIFF(day, '2022-01-01', '2022-01-05');
```

Ovaj upit će izračunati razliku između datuma 2022-01-01 i 2022-01-05 u danima, i rezultat će biti 4.

Pitanja iz priručnika za maturski ispit

216. Дата је табела ***RADNIK*** и упит:

IDBR	IME	PREZIME	PLATA	PREMIJA	DATZAP
6234	Marko	Pavlović	1300	3000	1990-12-17
6789	Janko	Nikolić	3900	10	1999-12-23

```
SELECT Ime, Prezime, DATEDIFF(year, DatZap, GETDATE()) AS God FROM
Radnik
```

Одредити шта је резултат упита. Заокружити број испред траженог одговора:

1. Табела са подацима о именима и презименима радника
2. Табела са подацима о именима, презименима и броју година које су протекле од датума запослења радника до краја века
3. Табела са подацима о именима, презименима и датумима запослења радника
4. Табела са подацима о именима, презименима и броју година које су протекле од датума запослења радника до тренутног датума

PODUPITI

Podupite u SQL-u su naredbe koje se koriste za filtriranje podataka u bazi podataka. Korišćenjem podupita, možete dobiti samo određene delove podataka koji zadovoljavaju određene uslove.

Primer podupita može izgledati ovako:

```
SELECT * FROM tabela WHERE uslov;
```

Ova naredba će izvući sve redove iz tabele koja zadovoljavaju uslov koji je naveden u WHERE klauzuli.

Primer uslova može izgledati ovako:

```
SELECT * FROM tabela WHERE ime = 'Tijana';
```

Ova naredba će izvući sve redove iz tabele u kojima se u koloni "ime" nalazi vrednost "Tijana".

Podupiti u SQL-u mogu biti vrlo korisni jer omogućuju da se vrlo precizno dođe do određenih podataka koji su potrebni za izveštavanje ili analizu podataka.

Podupiti koji vraćaju više vrednosti

Kada koristimo podupit koji vraća više vrednosti (tzv. multiple-row subquery), moramo koristiti operator IN, ANY ili ALL za poređenje vrednosti.

Operatori koji ne mogu biti korišćeni za poređenje sa multiple-row podupitima su:
Operator "=" (jednakosti) Operator "<>" (različitosti)
Operator ">" (veće)
Operator "<" (manje)
Operator ">=" (veće ili jednako)
Operator "<=" (manje ili jednako)
To je zato što ovi operatori očekuju jednu vrednost koju mogu uporediti sa drugom vrednošću, a multiple-row podupit vraća više vrednosti. U ovom slučaju, upotreba ovih operatora će izazvati grešku.

Na primer, ako imamo tabelu "Radnik" sa kolonama "Ime", "Prezime" i "Plata", i želimo da pronađemo sve radnike koji zarađuju više od prosečne plate, možemo koristiti multiple-row podupit ovako:

```
SELECT Ime, Prezime
FROM Radnici
WHERE Plata > (SELECT AVG(Plata) FROM Radnici);
```

Ovaj upit će izvući sve radnike čija je plata veća od prosečne plate u tabeli "Radnik".

Pitanja iz priručnika za maturski ispit

233 Заокружити бројеве испред тражених одговора.

Означити операторе који се **НЕ МОГУ** користити за поређење са подупитом који враћа више вредности:

1. ALL
2. ANY
3. BETWEEN
4. SOME
5. LIKE
6. IN

CONSTRAINTS

Constraints u SQL-u su pravila koja se primenjuju na kolone u tabelama kako bi se ograničio ili regulisao unos podataka u te kolone. Oni osiguravaju integritet podataka i sprečavaju unos nevažećih ili neprihvatljivih podataka.

Postoje različite vrste constraintsa koje se mogu primeniti na kolone u tabelama. Nekoliko najčešćih su:

NOT NULL - Ovaj constraint se primenjuje na kolonu kako bi se obezbedilo da ta kolona uvek sadrži validne podatke i ne može biti prazna.

Na primer:

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY,
  Ime VARCHAR(50) NOT NULL,
  Prezime VARCHAR(50) NOT NULL,
  GodinaRodjenja DATE NOT NULL,
  Adresa VARCHAR(100)
);
```

UNIQUE - Ovaj constraint se primenjuje na kolonu kako bi se osiguralo da ta kolona sadrži samo jedinstvene vrednosti i da ne postoje dve ili više identičnih vrednosti.

Na primer:

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY,
  Indeks INT UNIQUE,
  Ime VARCHAR(50) NOT NULL,
  Prezime VARCHAR(50) NOT NULL,
  GodinaRodjenja DATE NOT NULL,
  Adresa VARCHAR(100)
);
```

PRIMARY KEY - Ovaj constraint se primenjuje na jednu ili više kolona u tabeli kako bi se označile kao primarni ključ (primary key) tabele. Primarni ključ se koristi za jedinstvenu identifikaciju redova u tabeli.

Na primer:

```
CREATE TABLE Student (
  StudentID INT PRIMARY KEY,
  Indeks INT UNIQUE,
  Ime VARCHAR(50) NOT NULL,
  Prezime VARCHAR(50) NOT NULL,
  GodinaRodjenja DATE NOT NULL,
  Adresa VARCHAR(100)
);
```

FOREIGN KEY - Ovaj constraint se primenjuje na jednu ili više kolona u tabeli kako bi se označile kao strani ključ (foreign key) koji se odnosi na primarni ključ u drugoj tabeli. Strani ključ se koristi za uspostavljanje veze između dve tabele.

Na primer:

```
CREATE TABLE Ispit (
  IspitID INT PRIMARY KEY,
```

```
  StudentID INT,
  PredmetID INT,
  Ocena INT,
  FOREIGN KEY (StudentID) REFERENCES Studenti(StudentID),
  FOREIGN KEY (PredmetID) REFERENCES Predmeti(PredmetID)
);
```

CHECK - u SQL-u je ograničenje koje se koristi za provjeru da li novi ili ažurirani podaci u bazi podataka zadovoljavaju određene uvjete. U osnovi, CHECK constraint se koristi da se osigura da podaci u bazi podataka budu u skladu s određenim pravilima ili ograničenjima.

Primer CHECK constrainta može izgledati ovako:

```
CREATE TABLE Osoba (
ID INT PRIMARY KEY,
Ime VARCHAR(50),
Prezime VARCHAR(50),
DatumRodenja DATE,
Pol CHAR(1),
CONSTRAINT CK_Osoba_Pol CHECK (Pol IN ('M', 'Ž'))
);
```

U ovom primeru, CHECK constraint je korišćen za ograničavanje vrednosti koje se mogu uneti u kolonu "Pol" u tabeli "Osoba". Ograničenje se sastoji od uslova koji dopuštaju samo dva moguća unosa, "M" za muški pol i "Ž" za ženski pol. Dakle, ako pokušate uneti bilo koju drugu vrednost, kao što je "X" ili "Y", stvoriće se greška.

Korištenjem CHECK constrainta možete osigurati da se podaci u vašoj bazi podataka drže određenih pravila, što može pomoći u sprečavanju grešaka i održavanju integriteta podataka.

Constraints u SQL-u su korisni alati za osiguravanje integriteta podataka u tabelama. Primena odgovarajućih constrainta može pomoći u izbegavanju problema sa podacima i olakšati održavanje baze podataka.

Pitanja iz priručnika za maturski ispit

242 Заокружити бројеве испред кључних речи које се НЕ КОРИСТЕ за обележавање ограничења (*constraints*) у језику SQL:

1. Foreign key
2. Unique
3. Distinct
4. Check
5. Convert
6. Union
7. Not Null
8. Except

UNOS PODATAKA

INSERT INTO

INSERT INTO je SQL naredba koja se koristi za dodavanje novih redova u tabelu. Ova naredba se koristi zajedno sa VALUES klauzulom koja definiše vrednosti koje treba dodati u svaki red.

Sintaksa za INSERT INTO naredbu izgleda ovako:

```
INSERT INTO tabela (kolona1, kolona2, kolona3, ...)
VALUES (vrednost1, vrednost2, vrednost3, ...);
```

U ovoj naredbi, "tabela" predstavlja naziv tabele u koju se dodaju novi redovi, a "kolona1", "kolona2", "kolona3", itd. predstavljaju nazive kolona u koje treba uneti nove vrednosti. Nakon naziva tabela i kolona, VALUES klauzula sledi i definiše nove vrednosti koje se unose u svaki red.

Na primer, ako imamo tabelu "Student" sa kolonama "id", "ime" i "godine", naredba za dodavanje novog reda izgledala bi ovako:

```
INSERT INTO Student (id, ime, godine)
VALUES (1, 'Filip', 22);
```

Ova naredba bi dodala novi red u tabelu "Student" sa vrednostima "1" za kolonu "id", "Filip" za kolonu "ime" i "22" za kolonu "godine".

Takođe, važno je napomenuti da je moguće dodavati nove redove u tabelu bez navođenja naziva kolona, kao u primeru ispod:

```
INSERT INTO Student
VALUES (2, 'Tamara', 20);
```

Ova naredba bi dodala novi red u tabelu "Student" sa vrednostima "2" za kolonu "id", "Tamara" za kolonu "ime" i "20" za kolonu "godine", pretpostavljajući da se kolone u tabeli nalaze u istom redosledu kao u VALUES klauzuli.

INSERT INTO SELECT

INSERT INTO SELECT naredba u SQL-u koristi se za umetanje podataka iz jedne tabele u drugu. Ova naredba kombinuje INSERT INTO i SELECT naredbe i omogućava nam da jednostavno i efikasno preslikavamo podatke iz jedne tabele u drugu.

Na primer, ako želimo da prekopiramo podatke iz tabele "Student" u tabelu "novi_Studenti", možemo koristiti sledeću sintaksu:

```
INSERT INTO novi_Studenti (ime, godine, ocena)
SELECT ime, godine, ocena
FROM Student;
```

U ovom primeru, naredba INSERT INTO SELECT prvo navodi ciljanu tabelu "novi_Studenti" i kolone u koje će biti umetnuti podaci (ime, godine, ocena). Zatim koristimo SELECT klauzulu da bismo odabrali podatke koje želimo da prekopiramo iz tabele "Student". U ovom slučaju, želimo da prekopiramo podatke iz kolona "ime", "godine" i "ocena". Nakon toga, navodimo izvornu tabelu "Student".

Vrednosti koje se prenose mogu se kombinovati sa drugim vrednostima ili izrazima, npr.:

```
INSERT INTO novi_Studenti (ime, godine, ocena)
SELECT ime, godine + 1, ocena + 10
FROM Student
WHERE ocena > 5;
```

Ova naredba će umetnuti u tabelu "novi_Studenti" ime, godine i ocene svih studenata iz tabele "Student" čija je ocena veća od 5. Godine će biti uvećane za 1, a ocene za 10.

Važno je napomenuti da tabele u koje se podaci smeštaju i iz kojih se preuzimaju moraju imati iste kolone ili bar slične kolone koje se mogu preslikati jedna na drugu. Takođe, treba obratiti pažnju na redosled kolona, kako bi se podaci pravilno preslikali.

SELECT INTO

SELECT INTO je SQL naredba koja omogućava kreiranje nove tabele na osnovu podataka koji se nalaze u postojećoj tabeli.

Sintaksa naredbe SELECT INTO izgleda ovako:

```
SELECT kolona1, kolona2, ...
INTO nova_tabela
FROM stara_tabela
WHERE uslov;
```

kolona1, kolona2, ... su nazivi kolona koje želimo da prekopiramo u novu tabelu

nova_tabela je naziv nove tabele koju kreiramo

stara_tabela je naziv postojeće tabele iz koje želimo da prekopiramo podatke

Uslov je uslov po kojem se vrši selekcija podataka iz postojeće tabele

SELECT INTO naredba pravi novu tabelu sa istom strukturom kao i postojeća tabela. Razlika je u tome što nova tabela neće imati primarni ključ, strane ključeve i druge ograničenja koja se nalaze u staroj tabeli.

SELECT INTO naredba se može koristiti na različite načine. Na primer, možemo da izvršimo selekciju podataka iz više tabela i da ih prebacimo u novu tabelu. Takođe, možemo da izvršimo izračunavanja nad podacima i da ih prebacimo u novu tabelu.

Važno je napomenuti da SELECT INTO naredba nije standardna SQL naredba i da se može razlikovati u zavisnosti od baze podataka koju koristimo.

Pitanja iz priručnika za maturski ispit

234 Дата је табела ***Kupci*** са структуром:

```
( Id int primary key, Prezime varchar(50), Adresa varchar(50), Mesto
varchar(20), Telefon varchar(5), Status varchar(8) )
```

И табела ***NoviKupci*** са структуром:

```
( Id int primary key, Prezime varchar(50), Telefon varchar(20), Status
varchar(8) )
```

Извршава се упит:

```
INSERT INTO NoviKupci
SELECT * FROM Kupci WHERE Status <> 'Aktivan'
```

Одредити шта је резултат извршења датог упита. Заокружити број испред траженог одговора:

1. Како табела ***NoviKupci*** има све колоне које постоје и у табели ***Kupci,*** упит се извршава без грешке и у табелу ***NoviKupci*** се уписују записи из табеле **Kupci** са статусом који није **Aktivan**
2. Упит се не извршава, пријављује грешку јер се број колона у табели ***Kupci*** разликује од броја колона у табели ***NoviKupci***
3. Упит би се извршио без грешке да су у SELECT клаузули подупита, уместо * наведене све колоне табеле ***Kupci*** које имају своју одговарајућу колону у табели ***NoviKupci***
4. Упит јавља грешку због покушаја уписа вредности у поље примарног кључа који је аутоматски, тј креира га сама база

240 Креиране су табеле ***SKOLA*** и ***OSNOVNASKOLA***, а затим су у табелу ***SKOLA*** уписани подаци извршавањем следећих наредби:

```
create table Skola(
      skolaID int primary key, Naziv nvarchar(50),  gradID int, tip
nvarchar(50) )
create table OsnovnaSkola(
      gimID int primary key, Naziv nvarchar(50), gradID int )

insert into Skola values (101,'Nikola Tesla',20, 'srednja strucna')
insert into Skola values (102,'Dusko Radovic', 20,'osnovna')
insert into Skola values (103,'Sveti Sava', 30,'osnovna')
insert into Skola values (104,'Bora Stankovic', 20,'gimnazija')
```

У табелу ***OSNOVNASKOLA*** треба уписати податке о основним школама преписивањем потребних вредности из табеле ***SKOLA.*** Заокружити бројеве испред упита који ће јавити грешку при извршењу:

1. ```
   select * into OsnovneSkole
   from Skola where tip='osnovna'
   ```
2. ```
   insert into OsnovneSkole
   select * from Skola where tip='osnovna'
   ```
3. ```
   insert into OsnovneSkole(skolaID, Naziv)
   select s.skolaID, s.Naziv from Skola as s where tip='osnovna'
   ```
4. ```
   insert into OsnovneSkole
   select s.skolaID, s.Naziv, s.gradID from Skola as s where
   tip='osnovna'
   ```

246 Креирана је табела ***SKOLA***, а затим су у њу уписани подаци извршавањем следећих наредби:

```
create table Skola( skolaID int primary key, Naziv varchar(50) )

insert into Skola values (101,'Nikola Tesla')
insert into Skola values (102,'Mihajlo Pupin')
insert into Skola values (103,'ETS Zemun')
```

За дати упит, треба проценити сценарио који ће се десити и заокруживањем редних бројева испред исказа, означити могуће исходе:

```
select * into StrucneSkole from Skola
```

1. Креира се копија табеле Skola - нова табела под именом StrucneSkole исте структуре као и табела Skola и у њу се преписују сви подаци из табеле Skola
2. Уколико табела са именом StrucneSkole постоји у бази, креира се нова са именом StrucneSkole(1) и у њу се преписују сви редови из табеле Skola
3. Уколико табела са именом StrucneSkole постоји у бази, не креира се нова, само се у постојећу преписују редови из табеле Skola
4. Уколико табела са именом StrucneSkole постоји у бази, упит јавља грешку
5. Уколико се дода услов `where 1=2` упит се извршава, креира се нова табела исте структуре као и табела Skola, али се у њу не уписује ни један ред
6. Уколико се дода услов `where 1=2` упит јавља грешку јер је 1=2 увек нетачно тј. ***False***
7. Упит јавља грешку јер се кључна реч `into` користи искључиво у комбинацији са `insert`

POGLEDI

Pogledi u SQL-u su virtuelne tabele koje se sastoje od podskupa podataka iz jedne ili više tabela u bazi podataka. Oni su definisani SQL upitom i mogu se koristiti na isti način kao i obične tabele u SQL upitima.

Pogledi se koriste u situacijama kada je potrebno često izvršavati određeni upit na bazi podataka, a sam upit je kompleksan i dugotrajan. Tada se kreira pogled koji sadrži taj upit, i umesto da se izvršava kompleksni upit svaki put kada je potrebno doći do tih podataka, koristi se jednostavan upit nad pogledom.

Pogledi mogu biti i korisni kada je potrebno ograničiti pristup određenim delovima baze podataka za korisnike koji nemaju potrebna prava. Na primer, možete kreirati pogled koji prikazuje samo određene kolone iz tabele, a onda dozvoliti određenim korisnicima pristup samo tom pogledu, a ne celoj tabeli.

Kreiranje pogleda u SQL-u je jednostavno. Pogled se definiše kroz SQL upit, a zatim se kreira koristeći sintaksu CREATE VIEW. Nakon kreiranja pogleda, on se može koristiti na isti način kao i obična tabela u SQL upitima.

Pitanja iz priručnika za maturski ispit

225. Заокружити број испред одговора који представља наставак датог исказа:

Уколико поглед (view) треба користити за измену података у табели, поглед **НЕ СМЕ** садржати...

1. WHERE клаузулу
2. Спој више табела
3. Алијас колоне
4. GROUP BY клаузулу

226. Дат је упит за креирање погледа и наведени искази који се односе на дати упит. Заокружити број испред тачног исказа:

```
CREATE VIEW Pregled_Proseka AS
SELECT UcenikID, Ime, Prezime, AVG(Ocena) AS Prosek FROM Testovi
WHERE OdeljenjeID IN (1, 2, 3, 4)
GROUP BY UcenikID, Ime, Prezime
```

1. Подаци у табели ***Testovi*** се могу модификовати коришћењем погледа ***Pregled_Proseka***
2. Коришћењем датог погледа, подаци се могу само у додавати у табелу ***Testovi***, али не и мењати
3. Овако дат упит изазива грешку при извршењу
4. Коришћењем датог погледа, подаци из табеле ***Testovi*** се могу само прегледавати, али не и додавати или мењати