

Вовед во SQL

-SQL Претставува структуриран прашалнички јазик

SQL е јазик за дефинирање на податоци (**DDL**) и јазик за манипулирање со податоци (**DML**)

DDL се користи за да ги

>создаде (**create**)

>отстрани (**remove**)

>промени (**alter**)

описите во табелите (релациите) во базата на податоци

>CREATE SCHEMA

Одредува нова шема на база на податоци и и дава име. Оваа шема може да биде понатака променувана. Може да се дефинира и овластувачки идентификатор.

пр. CREATE SCHEMA KOMPANIJA AUTHORIZATION Jsmith;

>CREATE TABLE

Креирање нова релација со сите нејзини атрибути.

пр.

```
Create table department (  
name1 varchar(10) not null,  
unique number integer,  
primary key ssn char(9)  
);
```

исто така можат да се додадат и карактеристики на атрибутите како **primary key, not null, unique, foreign key + references**.

Надворешен клуч **foreign key (mngrSsn0) references emp(ssn)** може да биде дополнет со restrict, cascade, set null set default. Овие карактеристики се пишуваат после дефиницијата, можат да бидат on delete on update итн.

За секој атрибут може да биде поставена **default** вредност, и може да се одредат ограничувања со користење на **constraint**. Овие се проверуваат со клучниот збор **check**.

```
CREATE TABLE EMP (  
    ENAME          VARCHAR(30) NOT NULL,  
    ESSN           CHAR(9) ,  
    BDATE          DATE ,  
    DNO            INTEGER  DEFAULT 1  
        CHECK (DNO > 0 AND DNO < 21) ,  
    SUPERSSN       CHAR(9) ,  
    CONSTRAINT EMPSSN  
        PRIMARY KEY (ESSN) ,  
    CONSTRAINT EMPDEPT  
        FOREIGN KEY (DNO) REFERENCES DEPT(DNumber)  
            ON DELETE SET DEFAULT  
            ON UPDATE CASCADE ,  
    CONSTRAINT EMPSUPSSN  
        FOREIGN KEY (SUPERSSN) REFERENCES EMP(SSN)  
            ON DELETE SET NULL  
            ON UPDATE CASCADE );
```

Други типови во SQL:

>DATE: година-месец-ден

>TIME: час:минути:секунди

>TIME(I): исто како горе со дополнителен простор за секунди

>TIMESTAMP: Дата и време во едно

>INTERVAL: Релативна вредност, одредува интервал

>DROP TABLE -NAME- (cascade)

се користи за да се избрише некоја табела од базата. Доколку релацијата е референцирана во други, се користи Cascade

>ALTER TABLE

За промена на релации. По додавање новиот атрибут ќе има нулта вредност во сите торки, затоа не може да се додаде ненулта атрибут.

пр. Alter table employee add job varchar(12) or remove -name-

ПРАШАЛНИЦИ ЗА ПРЕБАРУВАЊЕ ВО SQL:

>SELECT:

Основна наредба за извлекување на информации од базата.

SQL дозволува да има торки кои се целосно исти. Табела е **вреќа од торки, не е множество**.

SQL релација може да биде ограничена да биде множество со користење на **PRIMARY KEY, UNIQUE**, за атрибутите или **DISTINCT** кога користиме прашалници.

Основна форма:

SELECT <ATTRIBUTE LIST> - Атрибути кои сакаме да ги извлечеме, пр. Име, Презиме итн..

FROM <TABLE LIST> - Листа на релации од кои сакаме да влечеме.

WHERE <CONDITION> -Логички израз кои треба да го задоволуваат торките, OR AND < > etc.

Овој основен вид на селект може да врати дупликатни торки.

>Со операторот **LIKE** може да се споредуваат делумни низи од знаци '%' или '*' заменуваат произволен број на знаци додека '_' заменува еден знак.

пр. Сите вработени родени во пеесетите

```
SELECT FNAME,LNAME
```

```
FROM EMPLOYEE
```

```
WHERE BDATE LIKE '____5_'
```

>Стандардни оператори *+/-/ може да се користат врз бројчани атрибути.

Доколку излистаме повеќе релации во FROM тогаш имаме пристап до сите нивни атрибути, и може да ги користиме за спојување на двете релации како што сакаме.

>Доколку две или повеќе релации имаат исти имиња за атрибути тогаш мора да го наведеме името на релацијата пред атрибутот кој го користиме, пр EMPLOYEE.BDATE.

Исто така може да се ставаат и прекари EMPLOYEE E or EMPLOYEE AS E

ПР.

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
```

```
FROM EMPLOYEE E S ИЛИ EMPLOYEE AS E, EMPLOYEE AS S
```

```
WHERE E.SUPERSSN = S.SSN
```

Доколку WHERE делот е испуштен тогаш нема услов. Во овој случај ако излистаме повеќе од една релација, во резултатот се добива декартов производ од релациите.

>Доколку ни требаат **сите атрибути** од една релација тогаш место нивните имиња може да го ставиме знакот '*'

**Q11: SELECT SALARY
FROM EMPLOYEE**

Q11: Salary
30000
40000
25000
43000
38000
25000
25000
55000

Q11A: Salary
30000
40000
25000
43000
38000
55000

**Q11A: SELECT DISTINCT SALARY
FROM EMPLOYEE**

>**DISTINCT** доколку сакаме да нема дупликати во резултатот

>**UNION, MINUS, INTERSECT** се вклучени во SQL со тоа што пресек и минус не во сите верзии.

Работат само на компатибилни релации и не враќаат дупликати.

>Во **WHERE** може да се користат и експлицитни множества со клучниот збор **IN**

```
SELECT DISTINCT SSN
```

```
FROM WORKS_ON
```

WHERE PNO IN (1,2,3) > Селектира социјално осигурување на работници што работат на проекти со број 1 2 3

>Во **WHERE** може да се вгнездат прашалници, се спојуваат со **IN**

```
SELECT FNAME
```

```
FROM EMPLOYEE
```

```
WHERE DNO IN (SELECT DNUMBER FROM DEPARTMENT WHERE DNAME = 'RESEARCH') Враќа број на вработени во research
```

>Доколку не користиме имиња на релации за имињата на атрибутите, тогаш тоа име се однесува на табелата во **најдлабокиот прашалник**. Прашалници поврзани со **IN** можат да се изразат како еден прашалник.

Ако вгнездениот прашалник користи атрибут од неговиот надпрашалник, тогаш се **меѓуповрзани**.

>**EXISTS** се користи за да се провери дали торката во внатрешниот прашалник постој. Доколку постој, тогаш и торката од надворешниот прашалник се враќа во резултатот.

пр.

```
SELECT FNAME FROM EMPLOYEE
FROM EMPLOYEE
```

```
WHERE EXISTS (SELECT * FROM DEPENDENT WHERE SSN = ESSN AND FNAME = DEPENDENT_NAME) Ќе ги прикаже
вработените кои имаат издржувани лица
```

Може да се користи и **NOT EXISTS**

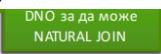
>За споредба со **NULL** се користи **IS** и **IS NOT**. '**=**' **Не работи**.

пр. SELECT FNAME FROM EPLOYEE WHERE SUPERSSN IS NULL < Вработени кои немаат надзорник

Ако има услов за спојување, торки со NULL не се враќаат.

>Во FROM делот може да имаме **споени** релации. Видови на JOIN кои се поврзани се JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN...

```
SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE
      NATURAL JOIN
      (DEPARTMENT AS
        DEPT (DNAME, DNO, MSSN, MDATE)))
WHERE DNAME='Research'
```



>АГРЕГАТНИ ФУНКЦИИ

> MAX, SUM, COUNT, MIN, AVG. Резултатот е секогаш **една** вредност и NULL вредностите се **занемаруваат**.

Прашалник 16. Најди ја најголемата, најмалата и просечната плата кај вработените кои работат во одделот 'Research'.

Прашалник 18. Врати го вкупниот бројот на вработени во одделот 'Research'.

```
Q16: SELECT MAX (SALARY), MIN (SALARY),
          AVG (SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
```

```
Q18: SELECT COUNT (*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
```

GROUP BY може да се користи доколку сакаме да ги групираме торките пред да се изведе врз нив агрегатна функција

Прашалник 20. За секој оддел, дај ги шифрата на одделот, бројот на вработени во тој оддел и нивната просечна плата.

```
Q20: SELECT DNO, COUNT (*), AVG (SALARY)
FROM EMPLOYEE
GROUP BY DNO
```

>**HAVING** делот се користи доколку сакаме агрегатната функција да се искористи врз групи кои исполнуваат одредени услови.

Прашалник 22. За секој проект на кој што работат повеќе од двајца вработени, вратете ги шифрата на проектот, името на проектот, како и бројот на вработени кои работат на тој проект.

```
Q22: SELECT PNUMBER, PNAME, COUNT (*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT (*) > 2
```

>**ORDER BY** делот се користи доколку сакаме да ги подредиме торките според некој услов. Нормалниот начин е по растечки редослед. (ASC DESC)

Прашалник во SQL може да има најмногу 6 делови, но само првите два, SELECT и FROM, се задолжителни.

Деловите од SQL прашалникот се пишуваат во следниов редослед:

SELECT	<attribute list>
FROM	<table list>
[WHERE]	<condition>
[GROUP BY]	<grouping attribute(s)>
[HAVING]	<group condition>
[ORDER BY]	<attribute list>

Ажурирање на бази на податоци

Постојат 3 наредби за промена на содржината на базата

>INSERT

>DELETE

>UPDATE

>INSERT

INSERT INTO <TABLE NAME> VALUES (<ATRIBUTES>)

Се користи за додавање на торки во табелата. Може да се додадат целосни или делумни торки.

Пр. INSERT INTO EMPLOYEE VALUES ('AAAA', 2, 'BBBB'...)

Вредностите на атрибутите мора да се запишуваат во истиот редослед како што биле дефинирани.

Може и експлицитно да се дефинира кои атрибути ќе ги внесуваме

INSERT INTO EMPLOYEE (NAME ,ADRESS...) VALUES (оаеииао)

Може и да се додаваат многу торки од еднаш кои се резултат на некој прашалник.

Пример: Сакаме да создадеме привремена табела која ќе ги содржи називите на одделите заедно со бројот на вработени и вкупните плати за секој оддел.

Доколку по извршување на овие два прашалници смениме нешто во табелите DEPARTMENT или EMPLOYEE во инфо

табелата ќе има стари информации, ова се разрешува со поглед

```
U3A: CREATE TABLE DEPTS INFO
      (DEPT_NAME VARCHAR(10) ,
       NO OF EMPS INTEGER,
       TOTAL_SAL INTEGER) ;
```

```
U3B: INSERT INTO
      DEPTS INFO (DEPT_NAME, NO OF EMPS, TOTAL_SAL)
      SELECT DNAME, COUNT(*) , SUM(SALARY)
      FROM DEPARTMENT, EMPLOYEE
      WHERE DNUMBER=DNO
      GROUP BY DNAME ;
```

>DELETE

DELETE FROM <TABLE> WHERE <CONDITIONS>

WHERE Содржи услови според кои ќе се бришат торките. Доколку WHERE не е дадено, тогаш се брише цела релација.

Се зачувува референцијалниот интегритет.

Избриши ги сите вработени што работат во одделот Research.

```
U4C: DELETE FROM EMPLOYEE
      WHERE DNO IN
      ( SELECT DNUMBER
        FROM DEPARTMENT
        WHERE DNAME='Research' )
```

>UPDATE

UPDATE <TABLE> SET <ATTRIBUTES> WHERE <CONDITIONS>

Променува вредности на атрибути.

UPDATE PROJECT

SET PLOCATION = 'иаоиаоаео', DNUM = 5

WHERE PNUM = 10 Ќе ги смени атрибутите на проектот со број 10.

НАПРЕДЕН SQL И ТЕХНИКИ НА ПРОГРАМИРАЊЕ НА БП

>ASSERTIONS (Општи ограничувања и тврдења)

>TRIGGERS (Активатори)

>VIEWS (Погледи)

>DB PROGRAMMING

>EMBEDDED SQL (Вградлив SQL)

>STORED PROCEDURES (Снимени процедури)

>ASSERTIONS

CREATE ASSERTION <IME> CHECK (USLOV)

Покрај проверки кои може да се дефинираат со ДДЛ, може да се дефинираат и дополнителни **тврдења**.

Пример. Вработените во некој оддел не може да имаат плата поголема од платата на шефот на тој оддел.

Проверуваме дали НЕ ПОСТОЈ таков човек.

Доколку не постој тогаш се е во ред.

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS
(SELECT *
FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
WHERE E.SALARY > M.SALARY AND
E.DNO = D.NUMBER AND D.MGRSSN = M.SSN))
```

>TRIGGERS

CREATE TRIGGER <IME> (AFTER|BEFORE) <NASTAN> ON <TABELA> [FOR EACH ROW] [WHEN <USLOV>] <AKCII>;

Ја мониторира базата на податоци и извршува одредени акции при одреден услов.

Синтаксата на активаторите ги вклучува следните делови:

➤ **Настан** <nastani>::=<nastan> {OR <nastan>}
➤ **Услов** <nastan>::=INSERT|DELETE|UPDATE [OF <kolona> {, <kolona>}]
➤ **Акција** <akcii>::=<PL/SQL block>

Пример. Да се напише активатор кој ќе ја споредува платата на вработениот со платата на неговиот шеф во текот на операциите внесување (Insert) и промена (Update)

```
EMPLOYEE
Fname Minit Lname Ssn Bdate Address Sex Salary Super_ssn Dno

CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
SALARY, SUPERVISOR_SSN ON EMPLOYEE
FOR EACH ROW
WHEN
(NEW.SALARY > (SELECT SALARY
FROM EMPLOYEE
WHERE SSN=NEW.SUPERVISOR_SSN)
)
INFORM_SUPERVISOR (NEW.SUPERVISOR_SSN, NEW.SSN);
```

CREATE TRIGGER TriggerName

<triggerTiming> <triggerEvent> [OR <triggerEvent>]

[OF <TableName.Attribute>[, <TableName.Attribute>, ...]]

ON <TableName>

[REFERENCING OLD as <alias_for_old_values>, NEW as <alias_for_new_values>]

FOR EACH {ROW | STATEMENT}

[WHEN (triggerCondition)]

BEGIN

<triggerBody>

END;

Деловите ограничени во < > се пополнуваат со конкретни вредности во зависност од барањата за тригерот.

Деловите ограничени во [] се опционални и може да се изостават.

<triggerTiming> = BEFORE | INSTEAD OF | AFTER

<triggerEvent> = INSERT | DELETE | UPDATE

<TableName> е табела за која се однесува <triggerEvent>

<TableName.Attribute> е конкретен атрибут од <TableName> за кој се однесува <triggerEvent>

OLD и NEW се објекти од типот <TableName> кој ги содржат вредностите пред да се изврши <triggerEvent> (старите вредности во OLD) односно после извршувањето на <triggerEvent> (новите вредности во NEW)

Тригерот може да биде **ROW или STATEMENT**. Подразбираната варијанта е **STATEMENT**.

Тригерот треба да биде **ROW** тригер доколку делот <triggerBody> треба да се изврши за секој ред од табелата <TableName> кој е засегнат од <triggerEvent>. Тригерот е

STATEMENT доколку се извршува само еднаш на ниво на табела доколку се случи

<triggerEvent> за табелата <TableName>.

WHEN делот се однесува на дополнителни услови кои треба да важат за да се извршат операциите од <triggerBody>.

>VIEWS

CREATE VIEW <IME> [<KOLONI>] AS <SELECT izraz>

Поглед е виртуелна табела изведена од други табели.

После AS е нормален прашалник. Креираната табела од поглед може да се користи понатака во други прашалници.

Кога оваа табела нема да ни е потребна може да искористиме **DROP <IME>** за да ја избришеме.

Материјализација на поглед е физичко креирање и чување на привремени табели. Се претпоставува дека ќе има и други прашалници над погледот. Доколку има разлики меѓу базната табела и погледот се случува **инкрементална промена (ажурирање)**.

Доколку погледот **не содржи агрегатни** функции, тогаш со негово ажурирање може да се ажурираат и базните табели.

Доколку вклучува соединувања, може да се ажурира, но не е секогаш возможно, затоа има **CHECK** опција.

Погледите добиени преку групирање и агрегатни функции не може да се ажурираат.

>CTE (COMMON TABLE EXPRESSION)

WITH expression_name [(column_name [,...n])]

AS

(CTE_query_definition)

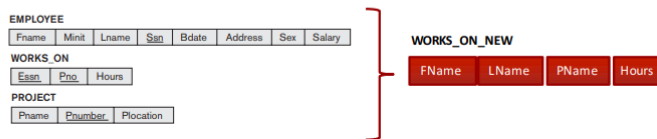
Се чува само дефиницијата, резултатот е за време на извршување. Помалце ресурси од поглед, Овозможува референцирање на резултатната табела повеќе пати во еден израз.

>Привремени табели

Се чуваат податоците, а не дефиницијата. Иста како обична табела само што се дефинира со # пред името. Треба да се избришаат после користење.

Пример. Специфицирај нова табела WORKS_ON_NEW во која ќе се чуваат податоците за името и презимето на учесникот во проектот, како и името на проектот и реализираните часови на истиот

**V1: CREATE VIEW WORKS_ON_NEW AS
SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE SSN=ESSN AND PNO=PNUMBER;**



ПРОГРАМИРАЊЕ НА БП

Базите на податоци многу често се пристапуваат од програми. Може да се случи да има некомпатибилности помеѓу базата и програмскиот јазик, како разлики во типовите. Исто така итератори за изминување на базата. Има неколку пристапи за програмирање.

>Вгнездени команди, SQL командите се вгнездуваат во програмскиот јазик.

>Библиотека од БП функции, достапна до програмскиот јазик позната како API.

>Сосем нов јазик

Чекори во програмирање кај БП: ESTABLISH CONNECTION > SEND QUERIES / RECEIVE INFO > TERMINATE CONNECTION.

- Променливите внатре во **DECLARE** се заеднички (се делат) и може да се појават во SQL изразите
- **SQLCODE** се користи за комуникација (errors/exceptions) помеѓу базата на податоци и програмата

```
int loop;
EXEC SQL BEGIN DECLARE SECTION;
    varchar dname[16], fname[16], ...;
    char ssn[10], bdate[11], ...;
    int dno, dnumber, SQLCODE, ...;
EXEC SQL END DECLARE SECTION;
```

```
loop = 1;
while (loop)
{
    prompt ("Enter SSN: ", ssn);
    EXEC SQL
        select FNAME, LNAME, ADDRESS, SALARY
        into :fname, :lname, :address, :salary
        from EMPLOYEE where SSN == :ssn;
    if (SQLCODE == 0) printf(fname, ...);
    else printf("SSN does not exist: ", ssn);
    prompt("More SSN? (1=yes, 0=no): ", loop);
    END-EXEC
}
```

>Вгнездување на SQL

Во повеќе програмски јазици може да се вгнезди SQL со одредени команди.

Тој се поставува меѓу изразите EXEC SQL (BEGIN) и END-EXEC или EXEC SQL END

Делените променливи добиваат префикс (:)

Итератор е потребен за обработка на повеќе торки.

Fetch го поместува итераторот на следната торка.

CLOSE CURSOR кажува дека обработката е завршена.

SQLJ, JDBC во јава.

>Динамички SQL

Создавање и извршување на нови (не компајлирани) SQL изрази за време на извршување.

>Снимени процедури

Функции кои се чуваат локално на серверот и може да бидат извршувани од повеќе програми.

Извршувањето на серверската страна намалува комуникациски трошоци.

Лошо е што секој СУБП има своја синтакса.

```
CREATE FUNCTION DSize (IN deptno INTEGER)
RETURNS VARCHAR [7]
DECLARE No_of_emps INTEGER;

SELECT COUNT(*) INTO No_of_emps
FROM EMPLOYEE WHERE Dno = deptno;

IF No_of_emps > 100 THEN
    RETURN "HUGE"
ELSEIF No_of_emps > 25 THEN
    RETURN "LARGE"
ELSEIF No_of_emps > 10 THEN
    RETURN "MEDIUM"
ELSE RETURN "SMALL"
ENDIF;
```

ФУНКЦИОНАЛНИ ЗАВИСНОСТИ И НОРМАЛИЗАЦИЈА

Дизајнот на една база на податоци има некои мерки за квалитет (групирање на атрибутите во “добри” релациски шеми).

Неформални мерки се:

Семантика – значење на атрибутите

Намалување на **редундантна** информација во торките

Намалување на бројот на НУЛЛ вредности

Недозволување да се создадат **лажни** торки

>Семантика:

Предлог 1: Неформално, секоја торка во релацијата треба да претставува еден ентитет или релациска инстанца од вистинскиот свет.

-Атрибутите од различни ентитети не треба да се мешаат во една релација.

-Само надворешни клучеви се користат за поврзување со други ентитети.

>Доколку податоци се чуваат **редундантно**, тоа троши дополнителен (непотребен) мемориски простор, И се појавуваат недостатоци при ажурирање на базата.

Предлог 2: Дизајн на шема која не подлежи на недостатоци при ажурирање, а доколку мора да ги има, треба да бидат означени.

Предлог 3: Бројот на торки со НУЛЛ вредности треба да се минимизира. Доколку постој некоја со многу НУЛЛ вредности за своите атрибути, тогаш таа може да се постави во посебна релација со референца.

>Лажни торки

Може да се појават како резултат на JOIN операција и лошо дизајнирана БНП.

Предлог 4: Запазувањето на правилото Lossless join, со гарантирање дека Join операцијата се врши врз пар примарен и надворешен клуч. Не смее да се појават лажни торки од природното соединување на било кои релации.

>Функционални Зависности (ФЗ)

Претставуваат формална мерка за добро дизајнирана БНП.

Заедно со клучевите се користат за дефинирање на **нормалните форми** на релациите.

Претставуваат **ограничувања** изведени од **значењето и меѓусебните** врски на податочните атрибути.

Се изведуваат од семантиката на реалните ограничувања на атрибутите.

>Множество на атрибути X функционално го одредува множество на атрибути T доколку вредност на X одредува единствена вредност за T.

Се прикажува со $X \rightarrow T$. Претставува **ограничување** над сите релациски инстанци $r(P)$.

За да важи мора да постојат две торки со иста вредност на X и тие мора да имаат иста вредност за T.

Во било која релациска инстанца $r(P)$ ако t_1 и t_2 имаат $t_1(X) = t_2(X)$ тогаш $t_1(T) = t_2(T)$.

>Од дадено множество на функционални зависимости може да изведеме и други со помош на правилата на изведување на **Армстронг**:

Рефлексија – ако T е подмножество на X тогаш $X \rightarrow T$

Множење - Ако $X \rightarrow T$ тогаш $XP \rightarrow TP$

Транзитивност – Ако $X \rightarrow T$ и $T \rightarrow P$ тогаш $X \rightarrow P$

Декомпозиција – Ако $X \rightarrow TP$ тогаш $X \rightarrow T$ и $X \rightarrow P$

Унија – Ако $X \rightarrow T$ и $X \rightarrow P$ тогаш $X \rightarrow TP$ (инверзно на горното)

Лажно пренесување – Ако $X \rightarrow T$ и $KT \rightarrow P$ тогаш $XK \rightarrow P$

Опсег на множество функционални зависимости Φ се означува со Φ^+ и ги зодржи сите ФЗ кои може да се изведат од Φ . Опсег на множество атрибути на X во зависност од Φ е множеството X^+ на сите атрибути функционално одредени од X. X^+ се пресметува со применување на Армстронг правилата.

>Две множества на ФЗ се **еквивалентни ако**:

Секоја ФЗ од M_1 може да се изведе од M_2

Секоја ФЗ од M_2 може да се изведе од M_1

значи ако $M_1^+ = M_2^+$

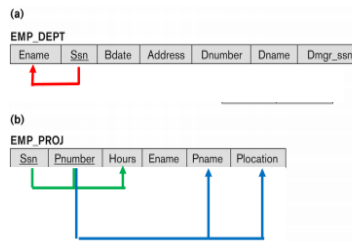
M_1 го **покрива** M_2 ако секоја ФЗ од M_2 може да се добие од M_1 . M_1 и M_2 се еквивалентни ако меѓусебно се покриваат.

>Множество Φ ФЗ е **минимално** доколку:

1. Секоја ФЗ има само еден атрибут од својата десна страна.

2. Не може да се промени $X \rightarrow T$ во $P \rightarrow T$ (каде P е подмножество на X) а новиото множество ФЗ да е еквивалентно со Φ .

3. Не може да отстраниме ФЗ од Φ и да добиеме множество еквивалентно со Φ .



➤ Матичниот број го одредува името на вработениот

SSN \rightarrow ENAME

➤ Шифрата на проектот ги одредува името и локацијата на проектот

PNUMBER \rightarrow {PNAME, PLOCATION}

➤ Матичниот број на вработениот и шифрата на проектот ги одредуваат неделните часови на вработениот за проектот на кој работи

{SSN, PNUMBER} \rightarrow HOURS

1. Множеството $F := E$.
2. Заменете ја секоја функционална зависност $X \rightarrow \{A_1, A_2, \dots, A_n\}$ во F со n-те функционални зависимости $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$
3. За секоја функционална зависност $X \rightarrow A$ во F, за секој атрибут B кој е елемент на X, ако множеството $\{F - \{X \rightarrow A\}\} \cup \{(X - \{B\}) \rightarrow A\}$ е еквивалентно на F, тогаш заменете ја $X \rightarrow A$ со $(X - \{B\}) \rightarrow A$ во F
4. За секоја преостаната функционална зависност $X \rightarrow A$ од F, ако множеството $\{F - \{X \rightarrow A\}\}$ е еквивалентно на F, тогаш отстранете ја $X \rightarrow A$ од F

Нормализација

Процес на декомпозиција на “лоши” релации со распарчување во помали релации.

Нормална форма е состојба која користи клучеви и ФЗ за да одреди дали релацијата е во нормална форма.

Се прави со цел да се добие визок квалитет на БП. Дизајнерите на БП немаат потреба да нормализираат до највисока нормална форма.

>Супер клуч на релациската шема $R = \{A_1, A_2, \dots, A_n\}$ е множеството атрибути S подмножество на R со својството дека не постојат две торки t_1 и t_2 во било која дозволена состојба r во R во која ќе важи $t_1[S] = t_2[S]$.

Исто така може да се дефинира како торка која го содржи клучот

>Клуч е супер клуч со дополнително својство дека со отстранување на кој било атрибут, веќе не е супер клуч.

Доколку релациска шема има повеќе клучеви, секој од нив се нарекува **кандидат клуч**.

Еден од нив се избира за **примарен клуч**, а останатите се **секундарни клучеви**.

Примарниот атрибут мора да припаѓа на еден од кандидат клучевите.

Форми:

Прва нормална форма 1нф:

Не дозволува:

>Композитни сложени атрибути.

>Повеќе-вредносни атрибути.

>Атрибути чии вредности за индивидуална торка не се атомични (вгнездени релации).



Normalization into 1NF.
(a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

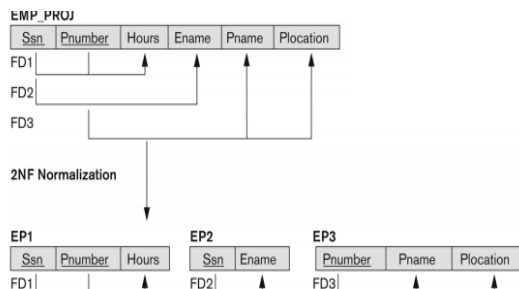
Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Втора нормална форма 2нф:

Релациска шема е во 2нф ако секој не-примарен атрибут А од Р е целосно функционално зависен од примарниот клуч.

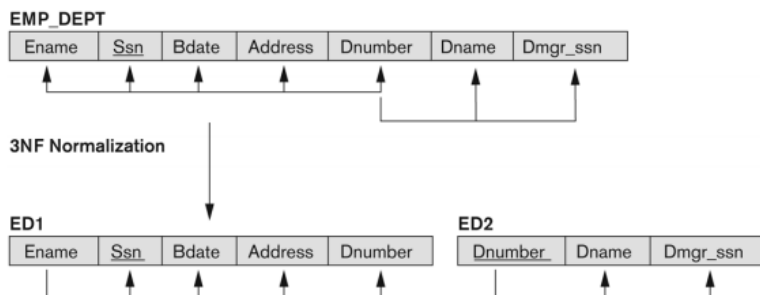
Нормализација во 2НФ

- $\{SSN, PNUMBER\} \rightarrow HOURS$ е целосна ФЗ се додека ниту $SSN \rightarrow HOURS$ ни $PNUMBER \rightarrow HOURS$ држат посебно
- $\{SSN, PNUMBER\} \rightarrow ENAME$ не е целосна ФЗ (се нарекува парцијална зависност) бидејќи $SSN \rightarrow ENAME$ исто така држи



Трета нормална форма 3нф:

Релациска шема Р е во 3нф доколку е во 2нф и нема не-примарен атрибут кој е транзитивно ($X \rightarrow T, T \rightarrow P$) зависен од примарниот клуч. Доколку е транзитивно зависен од преку кандидат клуч, тогаш нема проблем.



Нормална форма	Тест	Приод (нормализација)
Прва (1НФ)	Релациите не треба да имаат повеќевредносни атрибути или вгнездени релации.	Формирање на нови релации за секој повеќевредносен атрибут или вгнездена релација.
Втора (2НФ)	За релациите каде примарниот клуч содржи повеќе атрибути, неклучните атрибути не се функционално зависни од дел од примарниот клуч.	Разложување и поставување на нова релација за секој парцијален клуч со неговите зависни атрибути. Во релацијата да се задржи оригиналниот примарен клуч и сите атрибути кои се функционално зависни од него.
Трета (3НФ)	Релациите не треба да имаат неклучна атрибутна функционалност која е одредена од друг неклучен атрибут (или од множество на неклучни атрибути). Односно, треба да не постои преносна зависност на неклучен атрибут врз примарниот клуч.	Разложете ја и поставете релација која ги вклучува неклучните атрибути кои функционално ги одредуваат другите неклучни атрибути.

1ва нормална форма - Сите атрибути се зависни од клучот

2ра нормална форма - Сите атрибути се зависни од клучот во целост

3та нормална форма - Сите атрибути се зависни од ништо друго освен клучот

Генерални дефиниции

Релацииска шема е во 2нф ако секој не примарен атрибут е целосно функционално зависен од **секој** клуч на Р.

РШ е во 3нф ако постои ФЗ $X \rightarrow A$, каде **X** е **суперклуч** или **A** е **примарен атрибут**.

>Релација е во **Бојс кодова нормална форма (БКНФ)** Ако постои $X \rightarrow A$ тогаш **X** е **суперклуч**.

Сите 2нф се 1нф, сите 3нф се 2нф, сите **БКНФ(цел)** се 3НФ. Следната е многу појака од предходната.

ВОВЕД ВО ОБРАБОТКА НА ТРАНЗАКЦИИ (НЕ ТРЕБА)

Системи можат да бидат со еден корисник, или со повеќе корисници во еден момент.

Системот може да ги обработува процесите со **испреплетена обработка** (каде на еден процесор се извршуваат процесите) или **паралелна обработка** (каде процесите се извршуваат паралелно на повеќе процесори).

>**Транзакција** е логичка целина на обработка на базите која што вклучува една или повеќе операции за пристап. (read, write, update, delete).

Транзакцијата има **begin** и **end**.

>Упростен модел на БП

Збир на именувани податочни ставки, **Грануларност** на податоците, со основни операции **read** и **write**.

>read_item(X)

1. Find address of block on disk that has X
2. Put block on cache? Of main memory.
3. Copy X from cache to program variable X

>write_item(X)

1. find address of block on disk that has X
2. Put block on cache(меѓумеморија) of main memory
3. Copy X from program memory to cache
4. Save block from cache to disk.

>Lost update problem. Изгубено ажурирање

Кога **2 транзакции** истовремено пристапуваат **до исти податочни ставки**, така што **едната транзакција пребришува или заменува промена на друга транзакција**, правејќи ја конечната податочна ставка **грешна**.

T_1	T_2
read_item(X); $X := X - N;$	
write_item(X);	
read_item(Y);	read_item(X); $X := X + M;$
	write_item(X);
$Y := Y + N;$	
write_item(Y);	

Figure 21.3

Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.

Item X has an incorrect value because its update by T_1 is lost (overwritten).

>Temporary update problem (bad read). Привремено ажурирање

Транзакција ажурира податочна ставка, **но паѓа** поради некој проблем. Во овој случај, **транзакцијата ја враќа оригиналната вредност** на податочната ставка, **Но** во меѓу време, друга транзакција **веќе** ја **прочитала** **лошата вредност**.

T_1	T_2
read_item(X); $X := X - N;$ write_item(X);	
	read_item(X); $X := X + M;$ write_item(X);
read_item(Y);	

Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the temporary incorrect value of X.

>Неточно сумирање.

Се јавува доколку **една транзакција врши сумирање** на некоја податочна ставка, додека **друга ја ажурира**. Може да се случи да се сумираат некои стари и некои нови вредности, давајќи грешна сума.

>Неповторливо читање

Доколку една транзакција **чита 2 пати ист** запис, кој во меѓувреме е **сменет од друга транзакција**, давајќи **различни вредности**.

>Типови на прекини.

1. Пад на компјутер
2. Грешка во транзакција или систем
3. Локални грешки или исклучоци
4. Спроведување на контрола на конкурентност
5. Пад на диск
6. Физички проблем или природни катастрофи

БНП треба да може да се опорави од 1-4.

Транзакција претставува единица работа која се извршува **во целост** или не се извршува воопшто
Begin > read/write > end (проверка дали промените направени од транзакцијата може да се зачуваат).

Commit transaction – доколку транзакцијата завршила успешно.

Rollback (abort) – Неуспешно завршена транзакција, промените врати ги назад.