ИЗПИТНА ТЕМА 6 – Бази от данни

I. Бази от данни. Система за управление на бази от данни. Типове данни

Данните са съвкупност от събития или факти, описващи конкретно състояние на обект или група обекти в текущ момент от време. Те се предават, приемат (в блокове данни), съхраняват (във файлове, каталози, масиви, таблици, списъци в паметта на компютъра и се наричат операнди) и обработват.

Основни подхода за управление на данни:

- ✓ Традиционен - файлово-базиран подход, информацията се съхранява във вид на файлове.
- ✓ Бази от данни (БД).

1. Бази от данни - организирана съвкупност от данни в една предметна област, предназначена за съвместно използване от много потребители. БД е съвкупност от различни типове записи (файлове, данни) с определени отношения по между им, управлявани от СУБД.

В БД се прилагат следните отношения:

- „едно към едно“ (1:1), когато всеки запис от едно множество е свързан с един запис от второ множество и обратно.
- „едно към много“ (1:N), когато един запис от първата таблица (множество), се свързва с много записи от втората таблица, но запис от втората таблица се свързва само с един запис от първата таблица.
- „ много към едно “ (M:1) - вариант на „едно към много“. M:1 е най-често срещаният вид отношения между множества, при който един запис от първата таблица се свързва с много записи от втората таблица, но запис от втората таблица се свързва само с един запис от първата таблица.
- „много към много“ (M:N) - един запис от едната таблица се свързва с много записи от втората таблица, и обратно.

Моделите БД са:

- ✓ Класически модел БД
- ✓ Йерархичен модел - характеризира се с дървовидна структура. Той съдържа основен ствол и клони, като към всеки клон се добавят други малки клони. Данните са с йерархична връзка и всеки елемент е свързан единствено и само с един водещ елемент. Такъв тип връзка е “едно към много” (1:N).

Недостатък на йерархичният модел: един и същ логически запис може да се намира на няколко места в едно или в няколко дървета, като по такъв начин данните се дублират и се увеличава разходът на памет. За да се отстрани този недостатък се въвежда нов тип логически запис - виртуален, който съдържа указател към известен логически запис.

✓ Мрежови модел - ориентиран граф с върхове, които са множество от еднородни обекти и дъги, определящи връзките между тях. Тези модели се характеризират с функционален характер на връзките. БД при мрежовите и йерархичните модели се представят чрез записи и връзки между тях. Разликата между йерархичният и мрежовият модел се състои в това, че връзките в мрежовата структура могат да бъдат и от типа „много към много". Някои мрежови структури съдържат цикли, а по-сложните мрежови модели могат да се наблюдават и редица цикли. Когато едни записи от даден файл са свързани с други записи от същия файл се наблюдава т.н. пръстен. Пръстенът съдържа еднакъв тип записи, като породения запис съвпада с родителския запис. По такъв начин пръстенът е цикъл във файл с еднородни записи.

✓ Релационен модел - базиран на отношения между данни, таблично представени, като се постига еднородност, гъвкавост и достъпност на релационната структура.

Основни понятия в релационните БД:

Релацията е главна структура за моделиране на данните, структурата на БД е множество от релации.

Записът (кортеж, ред) е обект и информация за обекта.

Домейн е множество от неделими стойности, дефинирани логически; Доменът се дефинира с тип или формат на данните.

Например: датата - различен формат: уууу-mm-dd или dd mm, уууу

Релационна схема - крайно множество от имена на атрибути (колони). Броят на атрибутите се наричат степен на отношението. Данните в един атрибут са от един и същ тип.

Характеристики на релационните БД:

• Всяка таблица носи уникално име;

• Всеки атрибут (колона) има уникално име в рамките на таблицата;

• Всяка таблица съдържа уникални записи;

• Стойностите в записите са неделими, те са от един и същ тип данни, получени в резултат от изчисления.

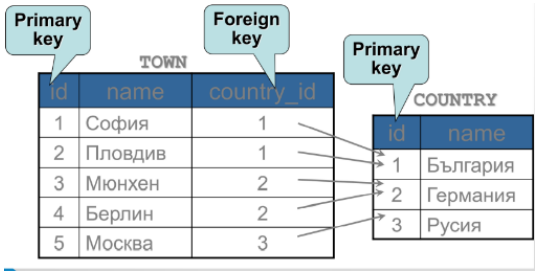
Релационният модел на данни се представя графично посредством диаграма на елементите и връзките, в която елементи са отделни таблици, а връзките между тях и съответните ключове са представени със стрелки, като се отчита типа на зависимостта: „едно към едно", „едно към много", „много към едно" „много към много".

Ключови елементи – уникални и еднозначни идентификатори на обекти. Правилният избор на ключов елемент води до достоверен концептуален модел.

Пример: Единният граждански номер носи информация за конкретен човек – име, презиме, фамилия, адрес, телефон, пол и т.н.

Съществуват няколко вида ключове:

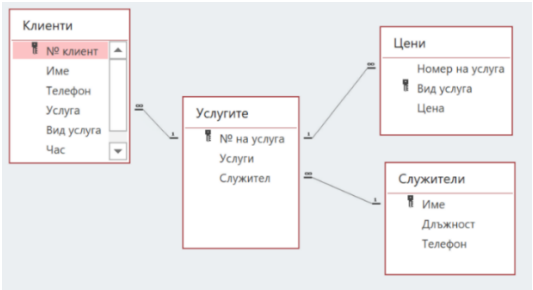
• **Първичен ключ** (primary key) (фиг. 1) – атрибут (колона) или група атрибути (колони), който идентифицира по уникален начин всеки запис. Често за първичен ключ се използва произволен уникален номер - данни от типа на автономериране. Два записа (реда) са различни когато са различни първичните им ключове. Връзките между таблиците се базират на взаимоотношения primary key / foreign.



Фиг. 1 Първичен ключ

• **Външен ключ** (foreign key) се използва за връзка между две и повече таблици. Той е необходим, за установяване отношения между две таблици (релации). Отношението се създава, като копие на първичния ключ на едната таблица се включи в структурата на втората таблица, за която той е външен (т.к притежава свой собствен първичен ключ);

На фиг. 2 е представена схемата на релационна на БД, изградена от четири таблици и съответните релации между тях.



Фиг. 2 Релационна схема БД

2. Система за управление на бази от данни (СУБД)

СУБД е набор от компютърни програми, контролиращи изграждането, поддръжката и използването на БД. СУБД предоставя на потребителя средство за описание на множеството от данните и е инструментът за манипулиране на данните вътре в БД.

Основни цели на СУБД:

- ✓ да ускори и улесни процеса на достъп до данните,
- ✓ да премахване и предотврати несъответствия и дублиране на данни,
- ✓ да подсигури сигурността на данните,
- ✓ да подпомогне обработката на голямо количество данни,
- ✓ да спести място за съхранение на данните.

СУБД се разделят на еднопотребителски и многопотребителски.

• Еднопотребителски (локални) са системи проектирани за работа с един потребител - Microsoft Access

MS Access -потребителски ориентирана програма за управление на БД, създадена за работа на персонални компютри и управление на малки БД. В Microsoft Access е възможно програмиране на език SQL.

Access е програма за управление на БД, в Windows, предоставяща различни възможности за въвеждане, съхранение, дефиниране, достъп и обработка на данните. С помощта инструменти се създават завършени и детайлно разработени приложения.

Таблица (Table) - обект за тематично записване на данни (задават се типове и характеристики на отделни обекти).

Заявка (Query) - извежда необходимата за потребителя информация, чрез селекция, сортиране, актуализация, и свързване на данни от различни таблици.

Форма (Form) - за улесняване при въвеждане и коригиране на данни. Те правят БД достъпна за потребителя.

Отчет (Report) - форми, готови за отпечатване. Те правят системата завършена.

Макрос (Makro) - последователност от действия (макрокоманди), с цел автоматизиране на дейности, извършени с клавиатурата и мишката в среда на Access, без програмен код.

• Многопотребителските системи за управление на бази данни са проектирани за работа от множество потребители. Фактори, които влияят върху броя на потребителите, които системата може да обслужи е оперативната RAM памет предоставена на системата за управление, мрежовата пропускливост и други.

Microsoft SQL Server - сървърна СУБД, изградена върху SQL (стандартен език за програмиране), създадена на базата на релационен модел на данни, предназначена за управление на големи, корпоративни БД. Тя поддържа голямо разнообразие от приложения за обработка на транзакции, бизнес разузнаване и анализи в корпоративни IT среди.

MySQL - многопоточна, многопотребителска, SQL СУБД (с над шест милиона инсталации), написан на езика C, разработена, разпространявана и поддържана от шведската компания MySQL AB. Потребители - NASA, Yahoo, Google, Amazon, LiveJournal, Cox Communications и др.

Oracle е мултимоделна БД. Тя поддържа БД с множество модели - документи, графики, релационни и ключови стойности. Oracle DB работи с Windows, UNIX, Linux и Mac OS.

3.Типове данни

Типът на данните е характеристика, посочваща каква информация може да се запише в дадена колона, параметър или променлива.

Типове данни в БД (Access):

Text (Short Text) - полета (с до 255 символа), съдържащи знаци, буквени, цифрови и символи. С числа представени, чрез този тип данни не могат да се извършват математически действия.

Пример: имена, телефонни номера (0878811199), ЕГН, пощенски кодове и др;

Memo (*Long Text*) - полетата (с до 64000 символа), които съдържат буквени и цифрови символи, бележки и резюмета. Те нямат стандартна дължина и не е възможно сортиране.

Number – числови полета (от 1-до 8 байта), които идентифицират даден обект (първичен или външен ключ) или подлежат на изчисления, те биват:

- *Byte* - цели числа (от 1 до 255 символа, 1 байт),
- *Integer* - цели числа от - 32768 до +32767 (2 байта, външен ключ) - могат да се подлагат на математически действия,
- *Long Integer* - цели числа от -2147483648 до 2147483647 (4 байта),
- *Single* -дробни числа с точност 0 до 6 знака (4 байта),
- *Double* - дробни числа с точност до 10 знака (8 байта),
- *Decimal* - дробни числа с точност до 18 знака (16 байта),
- *Date/Time* - полета използвани за съхраняване на дати и числа (8 байта),
- *Yes/No* - двоичен логически формат –True/False, Yes/No, On/Off (използва се само една от двете стойности (1 байт));
- *Currency* - полета използвани за съхраняване на разнотипни стойности (числа с точност до 15 знака, 8 байта). Съответният знак (валута, процент и т.н.) се записва автоматично според зададената стойност в контролния панел на Windows.
- *Auto Number* - уникален номер, който се задава автоматично за всеки нов запис. Този номер може да бъде последователен (да започва от 1), или произволен (4 байта). Тези полетата не могат да се променят.
- *OLE Object* - обект (графика, Excel диаграма или таблица, Word документ и др.), който е свързан или е вмъкнат в таблица, като се използва протокол OLE (до 1GB);
- *File* - логически свързана, именувана съвкупност от данни (информация), съхранявана върху запомнящо устройство.
- *Hyperlink* - поле, което съхранява връзка към друг обект в БД, друг файл или адрес в Web (до 64000 символа).

II. **Нормализация**

Нормализацията е формален метод за анализ на отношенията, с цел оптимизиране на релационния модел БД (икономия на памет и повишено бързо действие). Нормализация е набор от правила, чрез които релационните схеми се подлагат на проверки, за да се установи в коя от нормалните форми се намират. Критериите, определящи нивата на нормализация се наричат нормални форми.

1. Първа нормална форма (1NF) - позволява една стойност да се среща еднократно и никога като списък от стойности.

Правила:

- ✓ Във всяка клетка може да се запише само една стойност.
- ✓ Във всяка колона (поле) съдържа данни само от един и същ тип.
- ✓ Всяка колона в една таблица (поле) има уникално име.
- ✓ Последователността на записите и колоните е без значение.

2. Втора нормална форма (2NF)

Правила:

- ✓ Данните изпълняват 1NF.
- ✓ Не се допускат частични зависимости между колони в таблиците.

Пример: Таблица 1

StudId	SubjId	Mark	Teacher
1	1	4.5	Petrov
1	2	5.00	Ivanov
13	1	3.50	Petrov

В посочения пример в Таблица 1 полето "Teacher" е във функционална зависимост от полето "SubjId".

Решение на проблема: От посочената в примера таблица се премахват полетата "Teacher" и "SubjId" и се съставят две нови таблици, които съдържат „Teacher" и "SubjId", и с помощта на релации се осъществява желаната връзка.

3. Трета нормална форма (3NF)

Правила:

- ✓ Таблиците са в 2NF.
- ✓ Не се допуска транзитивна зависимост.

Транзитивност: Ако от **A** следва **B** и от **B** следва **C**, то тогата от **C** следва **A.**

Таблица 2

Id	**StudId**	**SubjId**	**Mark**	**Exam**	**Max Mark**
1	1	1	4.5	DB	6.00
2	1	2	90	Pract	100
3	13	1	3.50	DB	6.00

В посочения пример от Таблица 2 е налице транзитивна зависимост: "***Max Mark***" - „***Exam***" - „***SubjId***" и „***StudjId***", като последните две полета са ключови полета.

Решение на проблема: Премахват се полетата "***Max Mark***" и „***Exam***" от Таблица 2 и се съставят нова таблици с ключово поле „***StudjId"***, и полета "***Max Mark***" и „***Exam***".

4. Междинна нормална форма (3.5NF)

Правила:

- ✓ Таблиците да са в 3NF
- ✓ Полето (колоната), което не е първичен ключ не определя първичен ключ – т.е името на преподавателя не може да определя предмета, който преподава.

5. Четвърта нормална форма (4NF)

Правила:

- ✓ Таблиците са в 3.5NF.
- ✓ Да се разделят несвързаните данни

Таблица 3

Id	StudId	Exam	Hobby
1	1	DB	Music
2	1	Pract	Music
3	13	DB	Footbool

Пример: В Таблица 3 несвързани полета са „***Exam***“ и “***Hobby***”.

Изход: Да се създадат две таблици – за хобита и за курсове, като от Таблица 3 се премахнат колоните „***Exam***“ и “***Hobby***”.

6. Пета нормална форма (5NF)

Правила:

- ✓ Таблиците са в 4NF.
- ✓ Ако една таблица се декомпозирате до две таблици и след това се събере информацията отново в една таблица не се допуска да остане или да липсва информация извън таблицата.

7. Шеста нормална форма (6NF) - много краен вариант, не се прилага често.

Правила:

- ✓ Таблиците да са в 5NF.
- ✓ Всяка таблица се състои от първичен ключ и една колона.

III. Език SQL

SQL (Structured Query Language) - интегриран, структуриран език, за дефиниране и обработка на данни в релационни БД. SQL се вгражда в езици за програмиране от високо ниво - C, Pascal и др. Разновидности на SQL са PL/SQL, QL Procedural Language, Transact-SQL.

DDL – Data Definition language - команди като CREATE, DROP, ALTER

DML – Data Manipulation language - SELECT, INSERT, UPDATE, DELETE

DCL – Data Control Language - GRANT, REVOKE

Основните типове данни използвани в DDL, могат да се разделят на:

- Текстови - *char(size), nchar(size), varchar2(size), nvarchar2(size)*

- Числови данни - *number(p, s)*
- Дати - *date, timestamp*
- Други типове данни - *clob, nclob, blob, bfile, xmltype, Boolean*

1. Функции CRUD

CRUD:
"C" - "Insert",
"R" - "Select",
"U" - "Update",
"D" - "Delete".

✓ **Конструкция CREATE** служи за създаване на:

- База - *CREATE database dbemp*
- Таблица - *CREATE table temp*
- Изглед - *CREATE view vemp*
- Създаване на индекс - *CREATE index ixemp*

1.1 Създаване на таблица, добавяне и обновяване на стойности в нея:

```
CREATE TABLE TableName (column_name datatype,...),
INSERT INTO TableName VALUES (value(column_name), ...);
SELECT * FROM TableName;
```

Пример: Създаване на таблица „Groceries“ с три колони - „*ID*“ (първичен ключ), „*Name*“ и „*Quantity*“

```
CREATE TABLE Groceries (ID INTEGER PRIMARY KEY, Name TEXT, Quantity INTEGER );
INSERT INTO Groceries VALUES (1, "Apples", 4);
INSERT INTO Groceries VALUES (2, "Butter", 1);
INSERT INTO Groceries VALUES (3, "Chocolate", 3);
SELECT * FROM Groceries;
```

Резултат: Таблица 4

ID	Name	Quantity
1	Apples	4
2	Butter	1
3	Chocolate	3

Пример за изрична дефиниция на структурата на таблица 4 (без стойности):

```
CREATE TABLE people
(  ID INT NOT NULL,
  Name VARCHAR(30) NOT NULL,
  Quantity VARCHAR(20) );
```

Ограниченията в езика SQL спомагат за запазване интегритета на базата, за достоверност на данните и за неналичие на някои аномалии, т.е правят проверка за валидност на данните в определена колона

- NULL/NOT NULL – разрешава/забранява празни стойности
- UNIQUE – задължава една колона да има уникални стойности (връзка 1:1)
- DEFAULT – задава стойност по подразбиране.

Освен типа на дадено поле в таблица има и други характеристики, които задават условия, на които трябва да отговарят данните в дадено поле. Тези характеристики са известни като ограничения или ограничаващи фактори (constraint). Основна тяхна задача е да поддържат интегритета на данните - PRIMARY KEY, FOREIGN KEY, NOT NULL, CHECK, UNIQUE, DEFAULT.

В SQL има три операции за обновяване на БД, като всяка от тях важи само за една таблица.:

INSERT – въвежда редове в указана таблица.

DELETE – премахва редове от таблица. В случай, че не е зададено явно условие всички редове се изтриват и таблицата остава празна. Окончателното премахване на таблица от базата става с командата DROP.

Пример: Да се изтрият данните за продукт “Apples”.

```
DELETE FROM Groceries WHERE Name = 'Apples'
```

UPDATE – модифицира данни в таблица. С нея се променят стойности на избрани колони, удовлетворяващи определено условие. Имената на колоните, подлежащи на актуализация се указват след ключовата дума SET.

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Пример: Промяна на количеството “Apples” в Таблица 4.

```
UPDATE Groceries SET Name = 'Apples', 1
WHERE ID=1
```

В SQL оператор SELECT е оператор и за извличане на данни по критерии от БД.

1.2 Конструкция SELECT

SELECT се използва за извеждане на заявки – всяка SELECT конструкция се състои от две задължителни клаузи. Клаузата SELECT и клаузата FROM.

SELECT - завършва със символа; (точка и запетая). Например ако трябва да се извлече определена колона от таблица, то трябва да се използва конструкция от следния тип:

```
SELECT column_name(s)
FROM table_name;
```

Таблица – 5 “Persons”

ID	LastName	FirstName	Address	City
1	Ivanov	Ivan	Poduvo 7, apt.2	Sofia
2	Petrov	Petyr	Mladost 3, 358, b,	Sofia
3	Todorov	Dimitar	Prolet, 100	Kustendil

Пример: Да се изведат полетата „LastName“ и „FirstName“ от Таблица 5, “Persons”

SELECT LastName, FirstName
FPOM Persons;

Резултат: Таблица 6

LastName	FirstName
Ivanov	Ivan
Petrov	Petyr
Todorov	Dimitar

При използване на символ * се извеждат всички полета от таблица „Persons“.

*SELECT * FROM Persons*; (Таблица 5)

2. **Клауза WHERE**

WHERE се използва за извеждане на записи по определен критерии:

SELECT column_name(s)
FROM table_name
WHERE column_name operator value;

Пример*:* Да се изведат от таблица „Persons“ само живеещите в гр. София

*SELECT * FROM Persons*
WHERE City=’Sofia’;

Резултатът: Таблица 7

Id	LastName	FirstName	Address	City
1	Ivanov	Ivan	Poduvo 7, apt.2	Sofia
2	Petrov	Petyr	Mladost 3, 358, b,	Sofia

В клауза WHERE се използват следните оператори:

Оператори	Значение
=	Равно
< >	Различно
>; <	По-голямо; по-малко
>=; <=	По-голямо или равно; по-малко или равно
BETWEEN	Между
LIKE	Модул
IN	Избор

3. **Оператори за филтриране**

✓ AND (и двете условия са верни) и OR (поне едно от условия е вярно).

Пример: Да се изведат данните на хората с „LastName“ = “Ivanov” и „FirstName“ = “Ivan“ от таблица „Persons“.

*SELECT * FROM Persons*
WHERE LastName = ‘Ivanov’ AND FirstName = ‘Ivan’

Резултат: Таблица 7

Id	LastName	FirstName	Address	City
1	Ivanov	Ivan	Poduvo 7, apt.2	Sofia

Пример: Да изведат данните за хората с имена „FirstName“ = “Ivan“ или „Dimitar“ от таблица „Persons“.

*SELECT * FROM Persons*
WHERE FirstName = ‘Ivan’ OR FirstName = ‘Dimitar’;

Id	LastName	FirstName	Address	City
1	Ivanov	Ivan	Poduvo 7, apt.2	Sofia
2	Petrov	Petyr	Mladost 3, 358, b,	Sofia

✓ ORDER BY - сортиране във възходящ ред, за сортиране в низходящ ред се използва ключовата дума DESC.

SELECT column_name(s)
FROM table_name
ORDER BY column_name(s) ASC|DESC

Пример*:* Да се сортират данните от таблица „Persons“ в низходящ ред по колона „LastName“

*SELECT * FROM Persons*
ORDER BY LastName DESC;

Резултат: Таблица 8

ID	LastName	FirstName	Address	City
3	Todorov	Dimitar	Prolet, 100	Kustendil
2	Petrov	Petyr	Mladost 3, 358, b,	Sofia
1	Ivanov	Ivan	Poduvo 7, apt.2	Sofia

Пример: Ако са налице следните таблици:

„Employee“ с полета „FirstName“, „LastName“, „email“, „BirhtDate“, „Address“, „DepartmentNumber“, „Salary“, „Superssn“, “DNO”

„Department“ с полета „DepartmentNumber“, „DepartmentName“, „MGRSSN“

За да се изведе името на служител и номера и името на отдела в който работи се създава заявката:

SELECT Employee.FirstName, Employee.LastName, Department.DepartmentName
FROM Employee, Department
WHERE Employee.DepartmentNumber = DNO;

След оператора WHERE могат да се задават условия:

- за избор - DepartmentName = 'Research'
- за съединение – DepartmentNumber = DNO

В общия случай могат да се задават произволен брой условия за избор и съединение след оператора WHERE.

Пример: Да се изведат имената и адреса на служителите, работещи в отдел 'Research'.

```
SELECT Employee.FirstName, Employee.LastName, Employee.Address
FROM Employee, Department
WHERE Department.DepartmentName = 'Research' AND Department.DepartmentNumber= DNO;
```

В SQL е допустимо съществуването на две колони с едно и също име при условие, че те се намират в различни таблици.

Пример:

```
WHERE DepartmentName='Research' AND Department.DepartmentNumber= Employee.DepartmentNumber;
```

4. **Оператор LIKE** - за сравняване на части от низ или за търсене на определен подниз.

Символи: '%' - заменя произволен брой символи в низа,

Пример: Да се изведат имената на служителите, които живеят в 'Sofia,BG'

```
SELECT FirstNane, Lastname
FROM Employee
WHERE Address LIKE '%Sofi,BG%'
```

5. **Агрегатни функции в SQL**

- ✓ COUNT – брой на стойности в колона или брой на редове от таблица;
- ✓ SUM – сума от стойности на колона;
- ✓ MIN – най-малката стойност в колона;
- ✓ MAX – най-голяма стойност в колона;
- ✓ AVG – средно-аритметична стойност в колона.

Функциите COUNT, MIN и MAX могат да се прилагат върху числови или нечислови полета, докато SUM и AVG се прилагат само върху числови полета. Всички функции първо изключват стойностите Null, след което работят върху останалите стойности. Агрегатните функции се прилагат само върху една колона.

COUNT(*) - брой на редовете от таблица, независимо от наличните стойности Null. Агрегатните функции се използват само в оператора SELECT или заедно с ключовата дума HAVING.

Пример: Да се изчисли сумата от заплатите на всички служители, максималната и минималната заплата в предприятието.

```
SELECT SUM(Salary), MAX(Salary), MIN(Salary)
FROM Employee;
```

Пример: За всеки отдел да се изведат номерът му, броят на служителите и средната им заплата.

SELECT DNO, COUNT(*), AVG(Salary)

FROM Employee GROUP BY DNO

Пример: За всеки проект да се изведе номер, име и брой служители, работещи по него.

SELECT Project.ProjectNumber, Project.Projectname, COUNT(*)

FROM Project, WORKS_ON

WHERE Project.ProgjectNumber=PNO GROUP BY Project.ProjectNumber, Project.ProjectName

6. Вложени заявки

Основна характеристика на езика SQL - влагане на няколко SELECT-FROM-WHERE израза един в друг. Подобни заявки се формулират като вложени блокове след оператор WHERE на друга заявка, наречена външна заявка (outer query). Вложеният израз се наричат вътрешна заявка.

SELECT-FROM-WHERE SELECT FROM WHERE

Ако във външната и вложената заявки след FROM се срещат колони с еднакви имена, се използват синоними. По подразбиране неуточнена колона се свързва с таблицата, указана в най-вътрешния блок SELECT-FROM WHERE. Външната и вложената заявки са независими (некорелирани), когато във вложената заявка не се използуват имена на колони от таблицата, зададена във външната заявка. Ако във вложената заявка се използва колона, принадлежаща на таблица от външната заявка, двете заявки са свързани по между си и се наричат зависими *(корелирани)*.

Пример: Да се изведат имената и адресите на служителите от отдел 'Research'

SELECT FirstName, LastName, Address
FROM Employee
WHERE DNO IN
(SELECT DepartmentNumber FROM Department WHERE DepartmentName='Research')

6.1 Подзаявки

Подзаявката е оператор SELECT вложен в оператори - SELECT, INSERT, UPDATE, DELETE или в друга подзаявка. Подзаявката е заявка, чиито резултати се предават като аргумент на друга заявка. Подзаявките позволяват свързването на няколко заявки в една. Не могат да се използват подзаявки, обръщащи се към колони, съдържащи данни тип **text** и **image**.

Синтаксис:
SELECT column_name(s),…
FROM table_name1
WHERE table_name1 .column_name1 =
(SELECT < column_name1>
FROM < table_name2>
WHWRE < column_name2> = <values>)

6.2 Обединение – съединение между таблици и извеждане на заявка съдържаща елементи от няколко таблици - (INNER) JOIN

LEFT (OUTER) JOIN - Записи от лявата таблица и съответстващите записи от дясната

RIGHT (OUTER) JOIN - Всички записи от дясната таблица и съответстващите записи от лявата таблица

FULL (OUTER) JOINE - Всички записи, когато има съответствие в лявата или дясната таблица

Пример: Да се съставят, попълнят с по два записа в таблиците “Students” и “Student_grade”:

1. “Students” с колони: “ID” (първичен ключ),“FirstName”, ”LastName”, ” email”, ”phone”, “birthdate”

2. “Student_grade” с колони: “StudentID”,”test”,”grade”

```
CREATE TABLE Students (ID INTEGER PRIMARY KEY, FirstName TEXT, LastName TEXT, email
TEXT, phone TEXT, birthdate TEXT);
INSERT INTO Students (FirstName, LastName, email, phone, birthdate)
 VALUES ("Peter", "Petrov", "peter@abv.bg", "+3598888666", "2006-06-24");
INSERT INTO Students (FirstName, LastName, email, phone, birthdate)
 VALUES ("Alice", "Ivanova", "alice@gmail.com", ""+3598986764", "2009-07-04");

CREATE TABLE Student_grades (ID INTEGER PRIMARY KEY, StudentID INTEGER, test TEXT,
grade INTEGER);
INSERT INTO Student_grades (StudentID, test, grade)
VALUES (1, "Chemistry", 85);
 INSERT INTO Student_grades (StudentID, test, grade)
VALUES (2, "Chemistry", 95);
```

Пример за съединение: Да се състави заявка с която се изведат колоните “FirstName”, “LastName”, “email” и “grade” (релационна връзка между ID и StudentID и връзка M:1).

```
SELECT Students.FirstName, Students.LastName, Students.email, Students.test,
Student_grade.grade FROM Students
JOIN Student_grades
 ON Students.ID = Student_grades.StudentID
```

Резултат:

FirstName	LastName,	email	test	grade
Peter	Petrov	peter@abv.bg	Chemistry	85
Alice	Ivanova	alice@gmail.com	alice@gmail.com	95

Пълно съединяване – обединение на всички резултати, когато има съвпадение или в таблицата на дясно или в таблицата на ляво.

```
SELECT column_name(s) FROM table_name1 FULL OUTER JOIN table_name2 ON
column_name1=column_name2;
```

7. Транзакции в SQL

Транзакциите свързани с технически полета са начин за актуализиране на логическа единица информация в БД. Транзакция е въвеждането на една или повече промени в БД, като се групират множество SQL заявки и се стартира едновременно. Веднъж извършената транзакция не може да се върне назад.

SET autocommit = 0;

При използване на команди - CREATE, UPDATE, INSERT или DELETE, автоматично се стартира транзакция.

Ако се цели един UPDATE да се изпълни само когато се изпълни успешно друг, тогава се използват команди между BEGIN TRANSACTION и COMMIT:

Свойства на транзакциите (ACID)

- Атомарност (atomicity) - изпълняват се или всички, или нито една от модификациите, включени в транзакцията.
- Съгласуваност (consistency) - след приключване на транзакцията данните да останат логически верни. В една релационна БД към модификациите на транзакцията се прилагат ограничения, с цел поддържане цялостта на данните.
- Изолиране (isolation). Модификациите, които се извършват от едновременни транзакции са и автоматично изолирани.
- Дълготрайност (durability). При приключване на транзакция, резултатите от нейното изпълнение остават дори в случай на срив на системата. При срив на системата, докато транзакцията се изпълнява, тя се отменяне, без да оставя никакви частични следи върху данните. Например, ако преди да бъде завършена една транзакция, се получи прекъсване на захранването, когато системата ще бъде рестартирана. За да се осигури тази дълготрайност, се извършва предварително записване в дневника на транзакциите и автоматично отменяне и възстановяване на транзакциите при стартиране на SQL Server.