ЗМІСТ

Інформаційні системи

Згідно до стандарту ISO 2382-1 інформаційна система (ІС) — система обробки інформації, що працює спільно з організаційними ресурсами, такими як люди, технічні засоби та фінансові ресурси, які забезпечують і розподіляють інформацію. Інше визначення: ІС — сукупність апаратно-програмних та організаційних засобів для збереження та обробки інформації з метою забезпечення інформаційних потреб користувачів.

Основним завданням ІС є забезпечення конкретних інформаційних потреб у межах певної предметної області. Під *предметною областю* розуміють деяку частину реального світу. Переважна більшість сучасних ІС включають до свого складу бази даних та СУБД, тому на практиці часто синонімом до терміну ІС є термін «система баз даних».

Класифікація інформаційних систем

Виділяють кілька рівнів класифікації. По розподіленості:

1. настільні або локальні ІС (усі компоненти знаходяться на одному комп'ютері);
2. розподілені ІС:
   a. файл-серверні (БД знаходиться на сервері, СУБД та клієнтські програми — на робочих станціях);
   b. клієнт-серверні (БД та СУБД знаходяться на сервері, клієнтські програми — на робочих станціях).

По ступеню автоматизації ІС:

1. автоматизовані ІС (автоматизація неповна і необхідним є постійне втручання персоналу);
2. автоматичні ІС (автоматизовані).

По характеру обробки даних:

1. Інформаційно-пошукові (мета — пошук та видача інформації);
2. ІС обробки даних (АСУ та СППР).

По охопленню задач:

1. персональні;
2. групові;
3. корпоративні.

По сфері застосування:

1. навчальні;
2. військові;
3. економічні;
4. медичні;
5. географічні;
6. правові і т.п.

Бази даних

База даних (БД) — сукупність даних, яка відображає стан об'єктів деякої предметної області та зв'язки між ними і використовується для задоволення потреб користувача.

Базам даних притаманні наступні ознаки:

1. БД зберігається і обробляється у обчислювальній системі;
2. Дані у БД логічно структуровані (систематизовані), тобто виділені елементи, їх типізація та зв'язки між ними;
3. БД містить схему, яка описує логічну структуру БД.

По *моделі даних* БД класифікуються наступним чином:

- Ієрархічні — зображення БД у вигляді деревоподібної структури, що складається з об'єктів різних рівнів, між якими існують взаємозв'язки «предок-нащадок», причому у кожного об'єкта-нащадка є рівно один предок. Прикладом є файлова система та системи класифікації у біології.
- Мережеві — описується за допомогою орієнтованого графа, причому на відміну від ієрархічної моделі у вершину може входити кілька дуг та допускаються цикли. Прикладом є БД родинних зв'язків або транспортних систем.
- Реляційна — дані у БД є набором відношень, які задовольняють вимогам цілісності та підтримують оператори маніпулювання (реляційна алгебра).
- Об'єктно-орієнтована та об'єктно-реляційна — дані моделюються у вигляді об'єктів, їх атрибутів, методів та класів.

Системи керування базами даних

Система керування базами даних (СКБД) — сукупність програмних та мовних засобів, які забезпечують керування процесом створення та використання БД.

Найбільш розповсюдженими СКБД є Oracle, Interbase, IBM DB2, Informix, MS SQL Server, MySQL (клієнт-серверні); MS Access, Paradox, FoxPro (файл-серверні).

Основні функції СКБД:

- керування даними у зовнішній та оперативній пам'яті, забезпечення зберігання великих масивів даних;
- підтримка мов запитів та маніпулювання даними;
- фіксація змін даних, виконання транзакцій, забезпечення відновлення даних при збоях;

Склад сучасних СУБД:

- ядро (відповідає за керування даними у пам'яті);
- процесор мови БД (забезпечує оптимізацію виконання запитів та формування внутрішнього машинно-незалежного коду);
- підсистема підтримки часу виконання (інтерпретує команди маніпуляції даними);
- сервісні програми (утиліти).

Архітектура СКБД

Одним із найбільш важливих принципів побудови СКБД є ідея відокремлення логічної структури БД та маніпуляцій даними від фізичного способу реалізації.

По кількості рівнів описання системи розрізняють одно-, дво- та трирівневі системи. Трирівнева архітектура містить:

1. зовнішній (верхній) рівень, на якому користувачі сприймають дані;
2. внутрішній (нижній) рівень, на якому СКБД та операційна система сприймають дані;
3. концептуальний рівень, призначений для відображення зовнішнього рівня на внутрішній;

Описання структури даних на кожному рівні називається схемою. Головною метою трирівневої архітектури є забезпечення незалежності даних.

Концептуальний рівень СКБД. Сутності та зв'язки даних

Концептуальне проектування БД полягає у аналізі інформаційних потреб користувачів та визначенні потрібних їм елементів даних. Для опису *концептуальних схем* БД використовують моделі *"сутність-зв'язок"* (ER-модель). Концептуальна схема БД містить:

1. сутності та їх атрибути;

2. зв’язки між сутностями;
3. обмеження на дані;
4. семантична інформація про дані;
5. забезпечення безпеки та цілісності даних.

Сутності, атрибути та ключі

Сутність — це те, про що зберігається інформація.

Прикладом сутності є студент, працівник установи, клієнт, товар, замовлення.

Сутності описуються даними, які називаються *атрибутами*. **Атрибут** — це поіменована характеристика сутності, за допомогою якої моделюється деяка її властивість. Наприклад, клієнт як сутність описується номером, прізвищем, іменем, адресою та номером телефону. У БД фактично зберігаються лише атрибути. Значення кожного атрибуту вибирається із множини потенційних значень. Ця множина називається **доменом**.

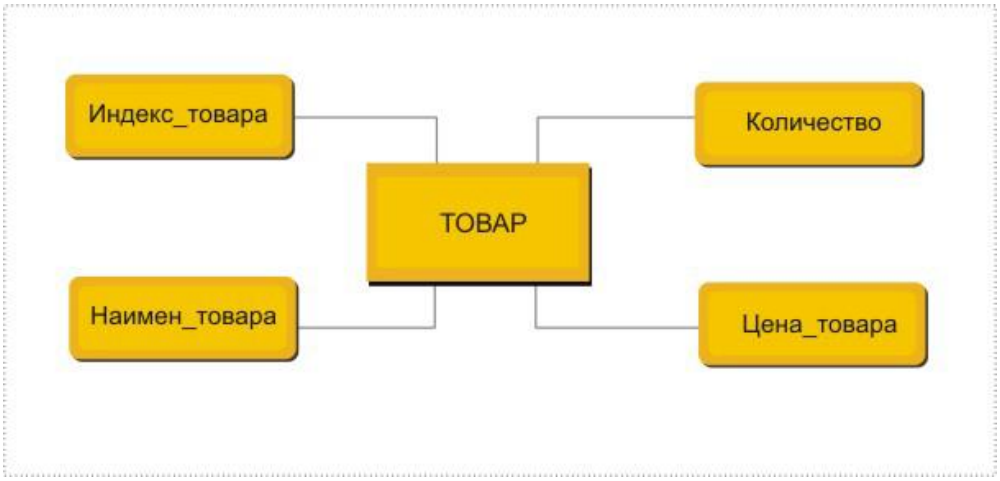

Рис. 1. ER-діаграма сутності ТОВАР та її атрибутів

Кожна група атрибутів, яка описує один об’єкт сутності, називається *екземпляром сутності*. Наприклад для сутності

СТУДЕНТ(ПІП, Група, Дата_народження)

екземпляром буде (Петренко, М31, 10.05.1994).

Серед атрибутів особливе положення такі, за допомогою яких можна ідентифікувати екземпляр сутності. Такі атрибути називають *ключами* або *індексами*. Потенційних ключів може бути кілька. Наприклад для сутності

ФАКУЛЬТЕТ(Код_факультету, Назва_факультету, ПІП_Декана)

ключовим може бути один з перших двох атрибутів.

Один із ключів вибирають у якості *первинного ключа*. Інші потенційні ключі називаються альтернативними. На ER-діаграмах первинні ключі підкреслюються. Іноді ідентифікацію проводять із використанням *складених ключів*, які містять кілька атрибутів. Наприклад сутність

ЛІКУВАННЯ(ПІП_лікаря, ПІП_пацієнта, Дата, Ліки)

однозначно ідентифікується складеним ключем із перших трьох атрибутів.

Зв’язки між сутностями

Дві сутності можуть бути з’єднані між собою. Кожному типу *зв’язку* присвоюється ім’я. Наприклад між сутностями ЛЕКТОР та ПРЕДМЕТ можна встановити зв’язок ВИКЛАДАЄ. Отримана структура сама по собі є сутністю, яка складається із пар екземплярів, взятих із обох сутностей та пов’язаних між собою. Тому сутність ВИКЛАДАЄ є складеною. На діаграмах сутність позначається наступним чином:

Рис. 2. ER-діаграма зв’язку ВИКЛАДАЄ

Рис. 3. ER-діаграма сутностей ГРУПА і СТУДЕНТ і зв’язку НАВЧАЄТЬСЯ

Під потужністю зв’язку розуміють максимальну кількість екземплярів однієї сутності, пов’язаних із одним екземпляром іншої сутності. Наприклад, потужність зв’язку ОДРУЖЕНІ рівна одиниці у кожному напрямку. Виокремлюють наступні типи зв’язку "один до одного" (1 : 1), "один до багатьох" (1 : *N*), "багато до багатьох" (*M* : *N*). Прикладом останнього типу зв’язку є зв’язок ВИКЛАДАЄ. Зв’язок (1 : *N*) існує між сутностями ГРУПА та СТУДЕНТ:

Зв’язок між сутностями *встановлюється з використанням атрибутів*. У попередньому прикладі у набір атрибутів сутності студент потрібно додати атрибут Код_групи.

Приклад моделювання локальної предметної області

У процесі моделювання предметна область (ПрО) розбивається на підобласті таким чином, щоб у них входило не більше 6-7 сутностей. Припустимо, що визначена ПрО: поставка товарів на склад. Форма поставки зображена на рис. 4.

Постачальник _____
Адреса постачальника _____
Індекс постачальника _____

Поставка

Дата поставки | Індекс поставки | № складу

Товари

Індекс	Найменування	Ціна	Кількість
……	……	……	……
……	……	……	……

Рис. 4. Форма поставки

Виділимо три сутності ПОСТАЧАЛЬНИК, ТОВАР ТА ПОСТАВКА. Із урахуванням типу зв’язків отримаємо наступну діаграму "сутність-зв’язок":

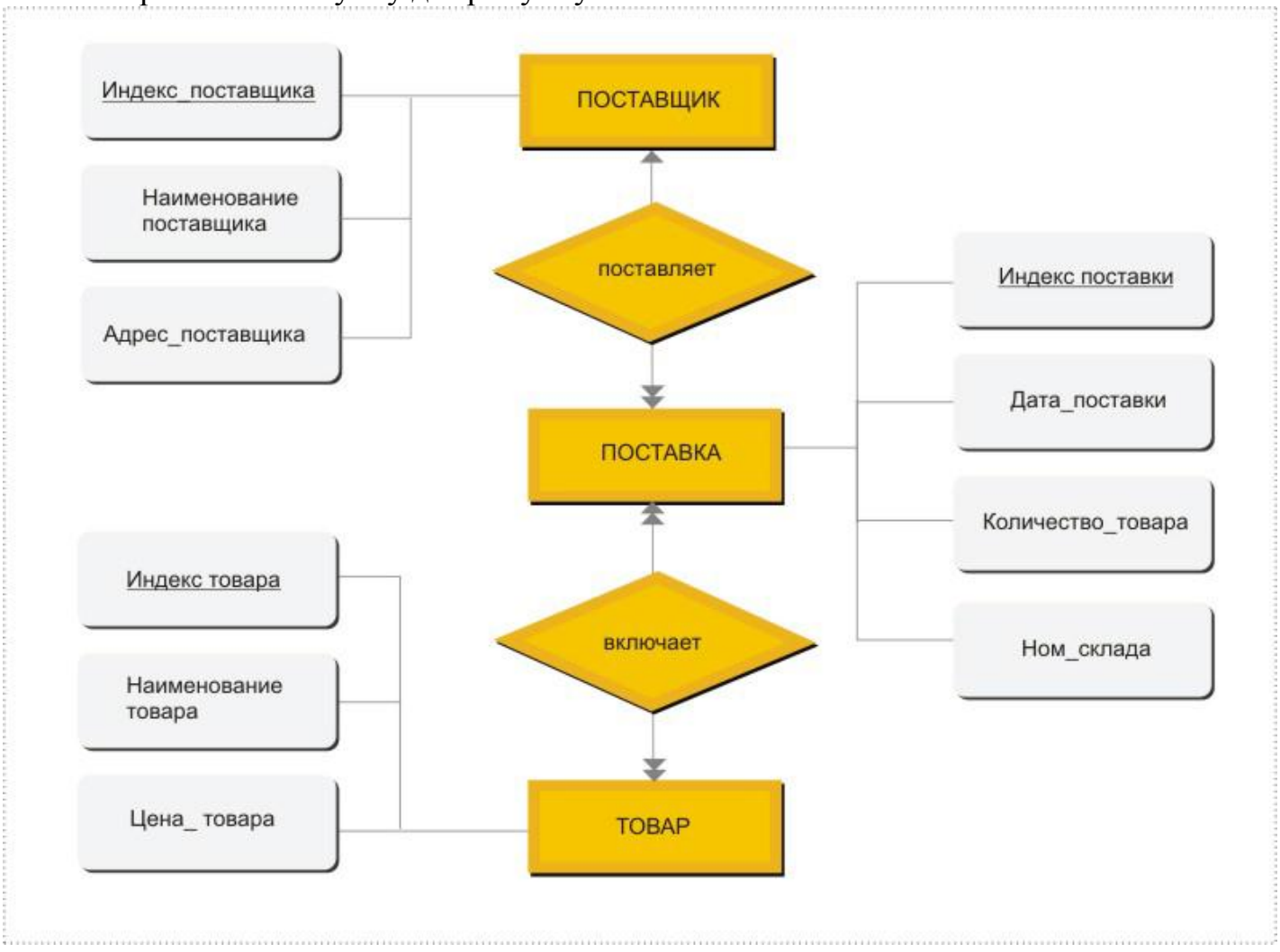

Рис. 5. Діаграма моделі предметної області поставка

Заключним кроком побудови концептуальної моделі ПрО є специфікація ключових атрибутів, визначення типів даних для атрибутів та специфікація зв’язків.

Реляційні СКБД

Реляційні СКБД засновані на *реляційній моделі* даних, розробленій співробітником IBM Е. Ф. Коддом. Основою реляційної моделі даних є теорія відношень. Усі дані зберігаються у *таблицях*. Кожний рядок таблиці має *однаковий формат*. Кодд сформулював дванадцять правил, яким має задовольняти допустима реляційна БД.

Структура реляційних БД

Реляційна база даних — це скінченний *набір відношень*. Відношення використовуються як для подання сутностей, так і для подання зв'язків.

Відношення — це двовимірна таблиця, яка має унікальне ім'я, рядки якої відповідають записам (екземплярам), а стовпці — атрибутам. Порядок слідування стовпчиків (атрибутів) не впливає на саме відношення.

Розглянемо приклад відношення для сутності

ВИКЛАДАЧ(Табельний_номер, ПІП, Посада).

Таблиця може мати наступний вигляд:

Табельний_номер	ПІП	Посада
1001	Шапочка І. В.	Завідувач кафедри
1002	Мич І. А.	Доцент
1003	Сливка-Тилищак Г.І.	Професор
……		

Нехай $D_1,\ldots,D_n$ — домени атрибутів відношення. Тоді відношення є *підмножиною декартового добутку* $D_1 \times \ldots \times D_n$. Елементи відношення називаються *кортежами*. Кожна компонента кортежів є *простим* (атомарним) значенням, тобто не складається із групи значень. *Кортежі мають бути унікальними*.

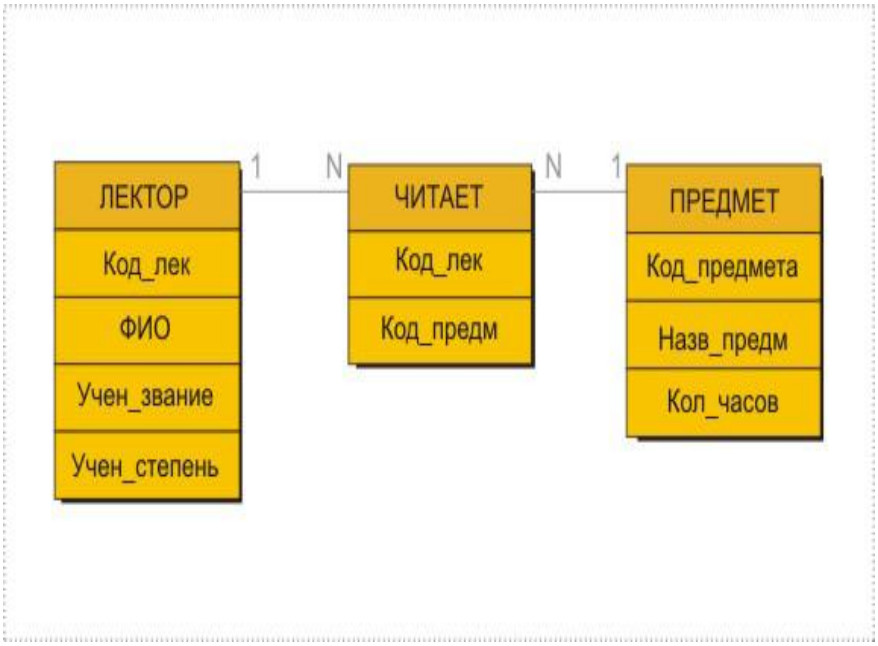

Рис. 6.

Реляційні ключі

Ключ — мінімальний набір атрибутів, значення яких повністю ідентифікують БД.

Зовнішній ключ — це набір атрибутів одного відношення, який є потенційним ключем іншого відношення. За рахунок зовнішніх ключів забезпечується можливість зв'язку між кортежами різних відношень. Відношення, яке містить зовнішній ключ, називається *дочірнім*, а відношення — яке містить відповідний зовнішньому потенційний ключ — *батьківським* (основним). Для моделі предметної області, зображеної на рис. 6 реляційна схема бази даних наведена на рис. 7.

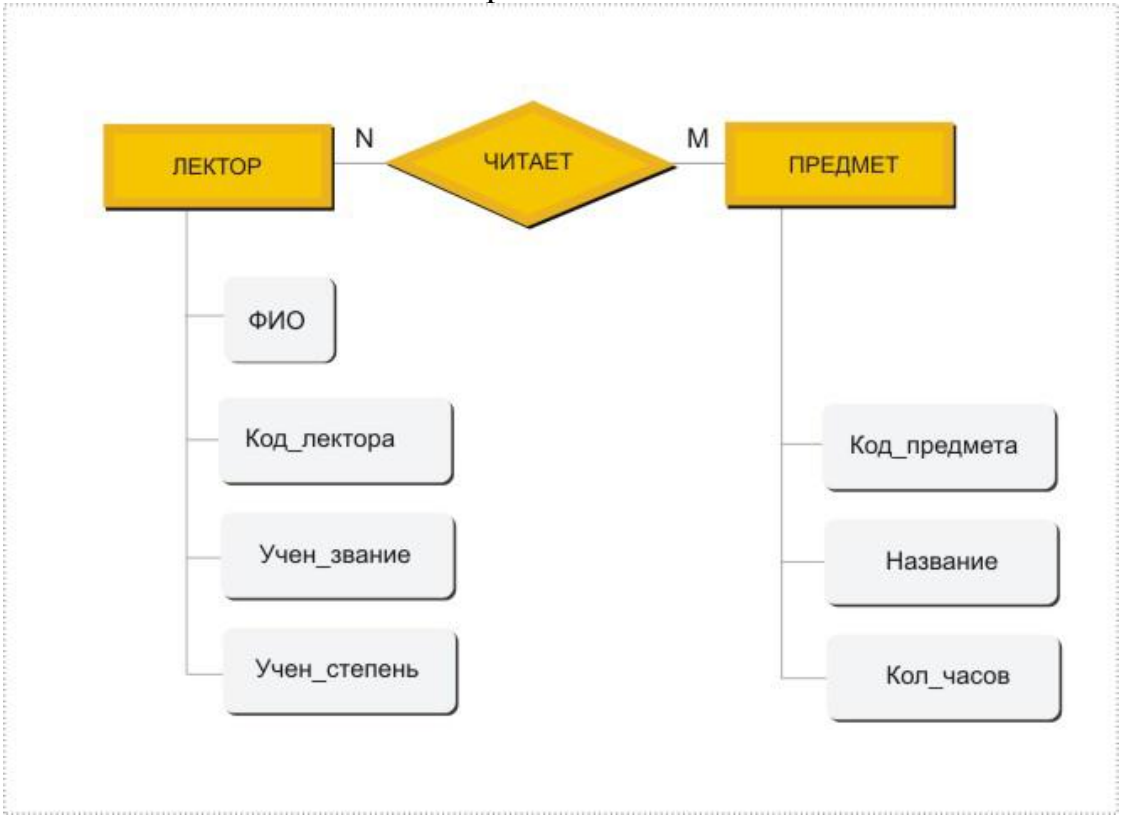

Рис. 7

Відповідні відношення мають наступні характеристики:

1. Лектор — 4-арне відношення з первинним ключем Код_лектора та доменами …
2. Предмет — тернарне відношення з первинним ключем Код_предмета та доменами …
3. Викладає — бінарне відношення із складеним первинним ключем (Код_лектора, Код_предмета).

Реляційна СКБД Microsoft Foxpro

Типи даних у FoxPro

Основні типи				Типи даних для полів			
Тип	Опис	Роз-мір	Пре-фікс	Тип	Опис	Розмір	Пре-фікс
Character	Символьний тип для зберігання тексту	1-254 байт	C	Double	Дійсне число з плаваючою крапкою	8 байт	B
Currency	Грошовий	8 байт	Y	Float	Синонім до Numeric	1-20 байт	F
Date	Дата	8 байт	D	General	Посилання на об'єкт		G
DateTime	Дата та час	8 байт	T	Integer	Цілий тип	4 байти	I
Logical	Логічний	1 байт	L	Memo	Посилання на блок текстової інформації		M
Numeric	Числовий (цілий або дробовий)	1-20 байт	N				

Команди для роботи з базами даних та таблицями

Команди для роботи з базами даних

1. Задання поточного каталогу

   `SET DEFAULT TO [cPath]`

2. Створення та відкриття (активація) БД:

   `CREATE DATABASE [Name | ?]`

3. Закриття поточної БД та всіх її таблиць (`ALL` — усіх БД):

   `CLOSE DATABASES [ALL]`

4. Закриття усіх відкритих файлів та активізація першої робочої області

   `CLOSE ALL`

5. Відкриття БД із заданим іменем

   `OPEN DATABASE [Name | ?]`

6. Задання поточної (активної) БД:

   `SET DATABASE TO [DatabaseName]`

7. Відкриття дизайнера таблиць для заданої БД

   `MODIFY DATABASE [DatabaseName | ?]`

Робота із таблицями

1. Створення таблиці з використанням дизайнера таблиць, добавлення її до поточної БД та активізація:

```
CREATE [TableName]
```

2. Створення таблиці із заданою структурою:

```
CREATE TABLE TableName
   (FieldName1 FieldType [(nFieldWidth [, nPrecision])] [PRIMARY KEY
   | UNIQUE]
   [NULL | NOT NULL]
   [, FieldName2 ...])
```

Довжина імені поля для первинного/унікального ключа **не повинна перевищувати 10 символів!**

3. Зміна структури поточної таблиці

```
MODIFY STRUCTURE
```

4. Добавлення вільної таблиці у поточну БД:

```
ADD TABLE [TableName]
```

5. Вилучення таблиці із поточної БД:

```
REMOVE TABLE [TableName] [DELETE]
```

Після вилучення таблиця стає *вільною* і, отже, може бути добавлена до іншої бази даних. Необов'язковий параметр DELETE вказує на те, що таблицю потрібно фізично видалити.

6. Перейменування таблиці

```
RENAME TABLE TableName TO NewTableName
```

Приклад.

```
CREATE DATABASE PostavkyTovariv
CREATE DATABASE MyDatabase
CREATE TABLE table1 (kod_tov I PRIMARY KEY, nazva_tov C(20), cina Y)
CLOSE TABLES && Закриття усіх таблиць у поточній БД
REMOVE TABLE table1
SET DATABASE TO PostavkyTovariv
ADD TABLE table1
RENAME TABLE table1.dbf TO Tovary.dbf
```

7. Показ інформації про таблиці активної БД

```
DISPLAY TABLES
```

8. Активація робочої області

```
SELECT nWorkArea | cTableAlias
```

Якщо параметр nWorkArea рівний нулеві, то активізується перша по порядку вільна робоча область. Функція SELECT() повертає номер активної робочої області.

9. Відкриття таблиці та її індексних файлів

```
USE [[DatabaseName.]Table|?] [IN nWorkArea | cTableAlias]
[INDEX IndexFileList] [ALIAS cTableAlias]
```

Відсутність опції IN вказує на активну робочу область. Якщо параметр *nWorkArea* рівний нулеві, то таблиця відкривається у першій по порядку вільній робочій області. Команда USE **не змінює активну робочу область.** Якщо у заданій робочій області вже є відкрита таблиця, то вона закривається.

10. Відкриття вікна Browse для перегляду та редагування активної таблиці

```
BROWSE [FIELDS FieldList] [FOR lExpression1]
```

11. Вивід вмісту активної таблиці

```
LIST [FIELDS FieldList]
  [Scope] [FOR lExpression1] [WHILE lExpression2]
```

Опція *Scope* може приймати наступні значення:

- All
- Next *Count*
- Record *Number*
- Rest

12. Створення нової таблиці на основі даних поточної таблиці

```
COPY TO FileName [DATABASE DatabaseName [NAME TableName]]
  [FIELDS FieldList] [Scope] [FOR lExpression1] [WHILE lExpression2]
```

13. Створення нової порожньої таблиці, яка має таку саму структуру, що і активна таблиця

```
COPY STRUCTURE TO FileName
  [FIELDS FieldList] [[WITH] CDX]
```

Індексування та сортування

1. **Індексування** (впорядкування записів поточної таблиці у порядку зростання значення індексного виразу)

```
INDEX ON eExpression TO IDXFileName | TAG TagName
[FOR lExpression] [DESCENDING] [UNIQUE | CANDIDATE] [ADDITIVE]
```

Опції DESCENDING та CANDIDATE доступні лише для мультиіндексних файлів.

2. Відкриття індексних файлів поточної таблиці

```
SET INDEX TO [IndexFileList | ? ]
  [ORDER nIndexNumber | IDXIndexFileName | [TAG] TagName] [ASCENDING
| DESCENDING]] [ADDITIVE]
```

Команда SET INDEX TO (без параметрів) діє аналогічно до команди CLOSE INDEXES, закриваючи усі індексні файли у активній робочій області (за винятком структурного мультиіндексного файлу).

3. Перебудова відкритих індексних файлів у поточній робочій області:

```
REINDEX
```

Приклад.

```
OPEN DATABASE PostavkyTovariv
```

```
CREATE TABLE Postavky (kod_postav I, data D, kod_postac I, kod_tov,
kilk i, sklad i)

BROWSE && Вводимо дані у таблицю

INDEX ON data to i1

SET INDEX TO && Вносимо зміни

SET INDEX TO i1 ORDER i1 DESCENDING

REINDEX

INDEX ON STR(sklad) + DTOC(data) to i2

LIST FIELDS sklad, data, kilk

SET INDEX TO
```

4. **Сортування** активної таблиці

```
SORT TO TableName ON FieldName1 [/A | /D] [/C]
[, FieldName2 [/A | /D] [/C] ...]
  [Scope]  [FOR  lExpression1]  [WHILE  lExpression2]  [FIELDS
FieldNameList]
```

Параметр /C вказує на те, що не потрібно враховувати регістр літер.

Переміщення по таблиці

1. Перехід на запис із заданим номером

```
GO nRecordNumber| TOP | BOTTOM [IN nWorkArea | cTableAlias]
```

2. Переміщення вперед або назад по таблиці

```
SKIP [nRecords] [IN nWorkArea | cTableAlias]
```

Пошук записів за критеріями

1. Послідовний пошук першого запису, який задовольняє задані умови

```
LOCATE FOR lExpression1 [Scope] [WHILE lExpression2]
```

Знаходження наступного запису, який задовольняє умови пошуку, вказані у LOCATE:

```
CONTINUE
```

Приклад.

```
SELECT Tovary
LOCATE FOR UPPER(LEFT(nazva_tov, 1)) = "A"
DISPLAY
CONTINUE
DISPLAY
```

2. Пошук з використанням індексу

```
SEEK eExpression  [ORDER nIndexNumber | IDXIndexFileName] [IN
nWorkArea | cTableAlias]
```

Для аналізу результатів пошуку можна використати функції RECNO() та FOUND().

Добавлення даних у таблицю

1. Добавлення запису у кінець таблиці

   ```
   APPEND [BLANK] [IN nWorkArea | cTableAlias]
   ```

   Покажчик активного запису переміщається на добавлений запис

2. Вставка запису

   ```
   INSERT [BLANK] [BEFORE] [IN nWorkArea | cTableAlias]
   ```

3. Добавлення у задану таблицю нового запису із заданими значеннями атрибутів

   ```
   INSERT INTO dbf_name [(fname1 [, fname2, ...])]
     VALUES (eExpression1 [, eExpression2, ...])
   ```

4. Добавлення у кінець активної таблиці записів із іншої таблиці

   ```
   APPEND FROM FileName      [FIELDS FieldList] [FOR lExpression]
   ```

5. Копіювання вмісту текстового файлу у мемо-поле

   ```
   APPEND MEMO MemoFieldName FROM FileName [OVERWRITE]
   ```

6. Імпорт OLE-об'єкту у general-поле

   ```
   APPEND GENERAL MemoFieldName FROM FileName [OVERWRITE]
   ```

Видалення та відновлення записів

1. Логічне видалення записів (позначення записів для видалення)

   ```
   DELETE
     [Scope] [FOR lExpression1] [WHILE lExpression2]
     [IN nWorkArea | cTableAlias]
   ```

   Приклад.

   ```
   SELECT Postavky
   DELETE FOR YEAR(data) <> 2014
   DELETE FOR DATE() – data > 21
   ```

2. Перевірка того, чи є активний запис позначеним для видалення

   ```
   DELETED()
   LIST FOR DELETED( ) && List marked records
   ```

3. Вибір режиму обробки записів, позначених як видаленні

   ```
   SET DELETED ON | OFF
   ```

   OFF (Default) — команди отримують доступ до логічно видалених записів

4. Фізичне видалення записів, позначених для видалення у активній таблиці

   ```
   PACK
   ```

5. Видалення усіх записів таблиці із збереженням її структури

   ```
   ZAP [IN nWorkArea | cTableAlias]
   ```

6. Відновлення записів активної таблиці, позначених як видалені

   ```
   RECALL
   ```

```
[Scope] [FOR lExpression1] [WHILE lExpression2]
```

Якщо діапазон та FOR не вказані, то команди DELETE та RECALL виконуються тільки для поточного запису

Модифікація записів

REPLACE FieldName1 WITH eExpression1 [, FieldName2 WITH eExpression2] ...
[Scope] [FOR lExpression1] [WHILE lExpression2] [IN nWorkArea | cTableAlias]

Приклад. Зменшити на 10% ціну останнього замовленого товару.

```
SELECT Postavky
GO BOTTOM
SELECT Tovary
REPLACE cina WITH cina*0.9 FOR kod_tovar = postavky.kod_tovar
```

Обчислювальні команди

1. Обчислення кількості записів активної таблиці

```
COUNT
   [Scope] [FOR lExpression1] [WHILE lExpression2] [TO VarName]
```

Приклад 1.

```
SELECT Postachalnyky
COUNT FOR AT("Ужгород", adresa) > 0 TO X
? X
```

Приклад 2. Вивід кількості дат поставок товарів.

```
SELECT Postavky
INDEX ON DATA TO i1 UNIQUE
COUNT TO X
? X
SET INDEX TO
```

2. Обчислення суми числових виразів, побудованих із полів активної таблиці

```
SUM [eExpressionList]
   [Scope] [FOR lExpression1] [WHILE lExpression2]
   [TO MemVarNameList | TO ARRAY ArrayName]
```

Приклад. Вивести сумарну кількість одиниць товарів, поставлених за останні 50 днів

```
SELECT Postavky
SUM Kilk FOR DATE() - data <= 50 to x
```

3. Обчислення середнього арифметичного числових виразів, побудованих із полів активної таблиці

```
AVERAGE [ExpressionList]
   [Scope] [FOR lExpression1] [WHILE lExpression2]
   [TO VarList | TO ARRAY ArrayName]
```

Приклад.

```
SELECT Tovary
AVERAGE CINA FOR nazva_tov = "moloko"
```

4. CALCULATE eExpressionList
   [Scope] [FOR lExpression1] [WHILE lExpression2]
   [TO VarList | TO ARRAY ArrayName]

   Список виразів може містити комбінацію наступних функцій:

   AVG(nExpression),
   CNT( ),
   MAX(eExpression),
   MIN(eExpression),
   STD(nExpression),
   SUM(nExpression),
   VAR(nExpression).

   Приклад 1. Вивести назви найдорожчих та найдешевших товарів

   ```
   CALCULATE MAX(cina), MIN(cina) to x, y
   BROWSE FOR cina = x OR cina = y
   ```

   Приклад 2. Вивести записи про поставки, найближчі по даті до 01.01.2014

   ```
   CALCULATE MIN(ABS(data - CTOD('01.01.2014'))) TO x
   BROW FOR ABS(data - CTOD('01.01.2014')) = x
   ```

5. Створення підсумкової таблиці, яка містить суми вибраних числових полів активної таблиці

   ```
   TOTAL TO TableName ON FieldName
     [FIELDS FieldNameList] [Scope]
     [FOR lExpression1] [WHILE lExpression2]
   ```

   Таблиця *повинна бути відсортована або проіндексована* по заданому полю!

   Приклад 1. Підрахувати по датам сумарну кількість одиниць поставленого товару

   ```
   SELECT Postavky
   INDEX ON data TO i2 FIELDS kilk
   TOTAL ON data TO Postavky2 FIELDS kilk
   SET INDEX TO
   USE postavky2 IN 0
   SELECT postavky2
   BROW FIELDS sklad, kilk
   USE
   ```

   Приклад 2. Для кожного складу підрахувати, скільки разів виконувалися поставки на цей склад.

   ```
   SELECT 0
   CREATE TABLE Postavky3 (sklad I, kilk I)
   APPEND FROM Postavky
   REPLACE kilk WITH 1 ALL
   INDEX ON sklad TO i2
   TOTAL ON sklad FIELDS kilk TO Postavky4
   SET INDEX TO
   USE Postavky4
   BROWSE
   USE
   ```

Об’єднання таблиць

```
JOIN WITH <Alias> TO <TableName>
  FOR <lExpression> [FIELDS <FieldsList>]
```

У таблицю-результат записуються записи, які відповідають парам записів активної таблиці та відкритої у іншій робочій області таблиці із заданим у команді псевдонімом, що задовольняють умову об’єднання. Якщо у відсутня опція FIELDS, то у таблицю-результат включаються всі поля обох таблиць.

Приклад. Створити та вивести таблицю, яка би містила році назву, ціну, кількість одиниць та дату поставок, зроблених у 2013 році.

```
OPEN DATABASE PostavkyTovariv
USE Tovary IN 0
USE Postavky in 0
SELECT Postavky
JOIN WITH tovary TO Postavky2013 FOR kod_tov =
tovary.kod_tov AND YEAR(data) = 2013;
  FIELDS tovary.nazva_tov, tovary.cina, kilk, data
USE postavky2013 IN 0
SELECT Postavky2013
BROWSE
USE
```

Зв’язок між таблицями

1. Зв’язок один до одного

```
SET RELATION TO
  [eExpression1 INTO nWorkArea1 | cTableAlias1 [, eExpression2 INTO
  nWorkArea2 | cTableAlias2 ...]
  [IN nWorkArea | cTableAlias] [ADDITIVE]]
```

Робоча область вказує місцеперебування дочірньої таблиці, яка має бути попередньо проіндексованою по атрибуту, по якому встановлюється зв’язок. Опція ADDITIVE вказує на те, що не потрібно розривати попередні зв’язки. При переміщенні покажчика активного запису основної таблиці покажчик активного запису дочірньої таблиці переходить на відповідний запис. Команда SET RELATION TO без параметрів розриває усі зв’язки у активній робочій області.

Приклад. Вивести коди поставок, назви відповідних товарів і постачальників та їх вартість (грошову суму, на яку було здійснено поставку).

```
SELECT Tovary
SET ORDER TO TAG kod_tov
SELECT Postachalnyky
SET ORDER TO TAG kod_postac
SELECT postavky
SET RELATION TO kod_tov INTO tovary, kod_postac INTO Postachalnyky
BROWSE FIELDS Postachalnyky.nazva_post, Tovary.nazva_tov, Vartist =
Tovary.cina*Postavky.Kilk
SET RELATION TO
```

2. Зв’язок один до багатьох.

```
SET SKIP TO [TableAlias1 [, TableAlias2] ...]
```

Попередньо треба встановити звичайний зв’язок за допомогою команди SET RELATION. При переміщенні по головній таблиці вказівник активного запису залишається на тій самій позиції поки вказівник запису дочірньої таблиці проходить через усі пов’язані записи.

Приклад. Для кожного товару вивести його назву та коди та кількість одиниць всіх його поставок.

```
SELECT Postavky
INDEX ON kod_tov TO i1
Select Tovary
SET RELATION TO kod_tov INTO Postavky
SET SKIP TO Postavky
BROWSE FIELDS nazva_tov, Postavky .kod_postav,
Postavky.kilk
SET SKIP TO
SET RELATION TO
```

Стандартні функції СКБД FoxPro

1. Функції для перетворення типів

```
STR(nExpression [, nLength [, nDecimalPlaces]])
VAL(cExpression)
CTOD(cExpression)
DTOC(dExpression)
CTOT(cExpression)
```

2. Функції для роботи з рядками

```
LEN(cExpression)
AT(cSearchExpression, cExpressionSearched [, nOccurrence])
LEFT(cExpression, nExpression)
RIGHT(cExpression, nCharacters)
SUBSTR(cExpression, nStartPosition [, nCharactersReturned])
LTRIM(cExpression)
RTRIM(cExpression)
ALLTRIM(cExpression)
LOWER(cExpression)
UPPER(cExpression)
PADL(eExpression, nResultSize [, cPadCharacter])
PADR(eExpression, nResultSize [, cPadCharacter])
```

3. Функції роботи з датою

```
DATE( )
DATETIME( )
YEAR(dExpression | tExpression)
MONTH(dExpression | tExpression)
DAY(dExpression | tExpression) && in month
```

DOW(dExpression | tExpression [, nFirstDayOfWeek]) && day of week (1 — sunday)

Команди SET CENTURY ON, SET DATE TO GERMAN

4. Функції, які показують положення покажчика активного запису

```
EOF([nWorkArea | cTableAlias])
BOF([nWorkArea | cTableAlias])
FOUND([nWorkArea | cTableAlias])
RECNO([nWorkArea | cTableAlias])
```

Функція IFF (функціональний IF)

```
IIF(lExpression, eExpression1, eExpression2)
```

Приклад. Відношення ПРАЦІВНИК(Код, ПІБ, Стать, Дата_народження, Посада, Оклад, Премія). Нарахувати жінкам премію у розмірі 500 грн., чоловікам — 300 грн.

REPLACE ALL PREMIA WITH IFF(Stat = "ж", 500, 300)

Нормалізація відношень

Нормалізація відношень — процес декомпозиції одного відношення відповідно до алгоритму нормалізації на декілька відношень на базі залежностей, які існують між групами атрибутів.

Нормальна форма (НФ) — властивість відношення в реляційної моделі даних, що характеризує його з точки зору надмірності. Нормальна форма визначається як сукупність вимог, яким має задовольняти відношення.

При роботі з відношеннями, які містять надмірні дані, можуть виникнути *аномалії* добавлення, видалення та модифікації кортежів.

Розглянемо відношення

СТУДЕНТ(Номер_зал, ПІБ, Група, ПІБ_старости, Куратор).

Аномалія добавлення виявляється при спробі створити нову групу і додати її у відношення, не вказавши при цьому інформацію про відповідного студента.

Аномалія видалення полягає у тому, що видалення запису про студента може призвести до видалення інформації про відповідну групу.

Аномалія модифікації полягає у тому, що при зміні інформації про групу (наприклад прізвища старости) треба модифікувати дані в усіх кортежах, у яких зберігається інформація про студентів цієї групи. Для усунення аномалій потрібно розбити початкове відношення на відношення

СТУДЕНТ(Номер_зал, ПІБ, Група)

та

ГРУПА(Група, ПІБ_старости, Куратор).

Перша НФ

Відношення знаходиться *у першій НФ*, якщо всі його атрибути приймають прості (атомарні) значення.

Приклад. Відношення

ПРАЦІВНИК(Код_працівника, ПІБ, Імена_дітей, Дати_народж_дітей)

не перебуває у першій НФ. Його потрібно розбити на відношення

ПРАЦІВНИК(Код_працівника, ПІБ),
ДІТИ(Код_працівника, Ім’я_дитини, Дата_народження).

Друга та третя НФ формулюються з урахуванням припущення про те, що відношення має єдиний потенційний ключ, який є одночасно первинним ключем.

Друга НФ

Відношення знаходиться *у другій НФ*, якщо воно знаходиться у першій НФ і не містить неключових атрибутів, які функціонально залежать від частини складеного первинного ключа. Відношення із простим первинним ключем завжди знаходиться у другій НФ.

Приклад. Відношення

ПОСТАВКА(Код_поставки, Номер_рядка, Код_товару, Назва_товару, Ціна, Кількість, Назва_Постачальника, Адреса_постачальника, Дата, Номер_складу).

Атрибути Дата, та Номер_складу функціонально залежать від коду поставки. Тому це відношення потрібно розбити на відношення

ПОСТАВКА(Код_поставки, Дата, Номер_складу)

та

ПОСТАВКА2(Код_поставки, Номер_рядка, Код_товару, Назва_товару, Ціна, Кількість, Назва_Постачальника, Адреса_постачальника).

Третя НФ

Відношення знаходиться *у третій НФ*, якщо воно знаходиться у другій НФ і не містить транзитивних залежностей (усі неключові атрибути функціонально залежать тільки від ключових атрибутів). Відношення із попереднього прикладу треба перетворити так:

ПОСТАВКА(Код_поставки, Дата, Номер_складу)

та

ДЕТАЛІ_ПОСТАВКИ(Код_поставки, Номер_рядка, Код_товару, Ціна, Кількість, Код_Постачальника),

ТОВАР(Код_товару, Назва_товару),

ПОСТАЧАЛЬНИК(Код_постачальника, Назва_Постачальника, Адреса_постачальника).

Нормальна форма Бойса-Кодда (НФБК)

НФБК формулюється для відношень із кількома складеними потенційними ключами, які перекриваються між собою. Відношення знаходиться у НФБК тоді і тільки тоді, коли детермінанти (аргументи) усіх його функціональних залежностей є потенційними ключами.

Приклад. Розглянемо відношення

ПОСТАВКА(Код_товара, Код_постачальника, Назва_постачальника, Кількість)

з унікальним атрибутом Назва_постачальника. Це відношення містить два потенційних ключі (Код_товара, Код_постачальника) та (Код_товара, Назва_постачальника). Атрибути Код_постачальника та Назва_постачальника визначають один одного, але вони не є потенційними ключами (а лише частинами ключів). Тому дане відношення не перебуває у НФБК. Для зведення до НФБК можна розбити початкове відношення на відношення

ПОСТАВКА(Код_товара, Код_постачальника, Кількість)

та

ПОСТАЧАЛЬНИК(Код_постачальника, Назва_постачальника).

НФБК дає змогу усунути аномалії оновлення даних, зумовленні функціональними залежностями.

Четверта НФ

Якщо кожному значенню атрибутів групи *A* відповідає множина значень атрибутів групи *B*, пов’язаних із *A* та множина значень групи атрибутів *C*, причому значення атрибутів груп *B* та *C* не пов’язані, то між атрибутами груп *A* та *B* (*A* та *C*) існує багатозначна залежність. Відношення знаходиться у четвертій НФ, якщо воно знаходиться у НФБК та не містить багатозначних залежностей крім, можливо, багатозначних залежностей від множини ключових атрибутів. Якщо відношення не знаходиться у 4НФ, то можливими є аномалії добавлення, видалення та модифікації.

Приклад. Відношення

ПРЕДМЕТ(Назва, Лектор, Підручник)

не знаходиться у 4НФ.

Мова запитів SQL

Мова *структурованих запитів* SQL є стандартною мовою реляційних СКБД. Крім запитів для отримання вибірок даних мова SQL містить підтримку *мови визначення даних* DDL (засоби для визначення структури БД) та *мови маніпулювання даними* DML (засоби для заповнення БД, модифікації та видалення записів). Для підтримки повноцінного інтерфейсу із БД у SQL передбачені функції:

1) створення, зміна та видалення таблиць, доменів, зображень, індексів, тригерів, збережених процедур та ін.;
2) підтримка цілісності та несуперечності БД;
3) захист даних шляхом визначення користувачів та їх ролей — прав доступу до даних та їх зміни;
4) маніпулювання даними у таблицях;
5) пошук у кількох таблицях та впорядкування даних;
6) підтримка транзакцій;
7) підтримка функцій та процедур користувача.

Є дві форми реалізації SQL. У *інтерактивному* SQL користувач вводить команди і безпосередньо отримує результат. Команди *вбудованого* SQL включаються до тексту програм на інших мовах програмування.

Типи даних SQL:

VARCHAR
BIGINT
SMALLINT
DECIMAL
BOOLEAN
TIMESTAMP
INTERVAL — проміжок часу
XML — дані XML.

Запити. Інструкція SELECT

Запит в SQL складається із однієї команди (інструкції) SELECT:

```
SELECT [DISTINCT | ALL] {* | список_виразів}
[INTO список_змінних]
FROM список_таблиць
[WHERE умова]
[GROUP BY список_виразів_групування
[HAVING групова_ умова]]
[ORDER BY список_виразів_упорядкування];
```

Основне призначення секції SELECT — вказівка стовпців таблиць, які потрібно вивести.

```
SELECT nazva_tov, cina
FROM tovary
```

Для виведення усіх стовпчиків таблиці у тому порядку, у якому вони йдуть у таблиці, використовується символ *.

```
SELECT *
FROM tovary
```

Ключове слово DISTINCT вказує на вилучення однакових рядків у результаті запиту

```
SELECT DISTINCT nazva_tov
FROM tovary
```

Альтернативою до DISTINCT є ключове слово ALL, яке використовується по замовчуванню.

Інструкція SELECT може містити не тільки стовпчики таблиці, але й вирази. Крім того можна вказувати назви стовпців результату та включати текст у результат запиту:

```
SELECT 'Товар', nazva_tov AS 'назва', cina*1.1 AS 'націнка'
FROM tovary
```

У списку таблиць фрази FROM можуть використовуватися синоніми.

```
SELECT t.cina
FROM tovary AS t
```

Умова відбору рядків. Секція WHERE

```
SELECT kod_tov, cina
FROM tovary
WHERE LOWER(nazva_tov) = 'молоко'
```

Зауваження. У MS Access для перетворення регістру рядків замість передбачених стандартом функцій LOWER та UPPER використовуються функції LCASE та UCASE.

```
SELECT *
FROM postavky
WHERE data < '2014-01-01' AND NOT(sklad = 2)
```

Зауваження. Форма запису літералів типу дата у різних СКБД може бути різним. У попередньому прикладі використано міжнародний формат запису дати, який далеко не завжди "розуміють" сучасні СКБД. Наприклад, у MS Access замість одинарних лапок використовуються символи # і відповідна константа запишеться так: #2014-01-01#.

У фразі WHERE можна використовувати спеціальні перевірочні предикати:

- IN (список значень) — належність до множини;
- IS NULL — значення невизначене;
- BETWEEN мінімум AND максимум — потрапляння у заданий замкнений діапазон;
- LIKE шаблон — відповідність шаблону;

Приклади.

```
SELECT *
FROM postavky
WHERE sklad IN (1, 3, 4)

SELECT *
FROM tovary
WHERE cina BETWEEN 20 AND 50

SELECT *
FROM postavky
WHERE NOT(data IS NULL)
```

Предикат `LIKE` виконує порівняння значення рядкового виразу та множини значень, які визначаються *шаблоном*.

Шаблон — це рядок, який разом із звичайними символами може містити *символи підстановки* % та _.

Символу % відповідає довільна послідовність символів (у тому числі і порожня).

Символу _ відповідає довільний один символ.

Наприклад, шаблону 'Іван%о%' відповідає 'Іваненко', 'Іванченко', 'Іванова', 'Іван Петров' (але не відповідає 'Іванишин'), а шаблону 'л_с_' — 'лист', 'лоск', але не відповідає 'лісний', 'ласка' тощо.

```
SELECT *
FROM postachalnyky
WHERE adresa LIKE '%м. Київ%'
```

```
SELECT *
FROM tovary
WHERE nazva_tov NOT LIKE 'молочн%'
SELECT *
FROM tovary
WHERE nazva_tov LIKE 'c___'
```

Багатотабличні запити

Інструкція SELECT надає можливість виконувати з'єднання таблиць, результатом якого є кортежі, які містять значення вказаних атрибутів таблиць, що беруть участь у з'єднанні. Найпростішим типом з'єднання є декартів добуток таблиць.

```
SELECT * FROM tovary, postavky
```

Умова з'єднання вказується у секціях WHERE або FROM. Найчастіше таблиці з'єднуються з використанням умови рівності.

З'єднання у секції WHERE

```
SELECT t.nazva_tov, t.cina
FROM tovary t, postavky p
WHERE t.kod_tov = p.kod_tov AND p.kilk > 100
```

```
SELECT t.nazva_tov, p.nazva_post
FROM tovary t, postavky, postachalnyky p
WHERE t.kod_tov = postavky.kod_tov AND
                              p.kod_postac = postavky.kod_postac
```

Можна також з'єднувати таблиці з використанням нерівностей та виконувати *самоз'єднання*.

Приклад. Вивести назви товарів, ціна яких перевищує мінімальну ціну більше ніж у двічі.

```
SELECT DISTINCT t1.nazva_tov
FROM tovary t1, tovary t2
WHERE t1.cina > 2*t2.cina
```

Приклад. Вивести дати, у які було зроблено не менше двох поставок на склад №1.

```
SELECT DISTINCT p1.data
FROM postavky p1, postavky p2
WHERE p1.data = p2.data AND p1.kod_postav <> p2.kod_postav AND
                                    p1.sklad = 1 AND p2.sklad = 1
```

З'єднання у секції FROM

```
Таблиця [INNER | {FULL | LEFT | RIGHT} [OUTER]] JOIN Таблиця ON умова
```

За замовчуванням виконується внутрішнє з'єднання.

Зауваження. У Microsoft Access ключовому слову JOIN *обов'язково* мають передувати ключові слова INNER, LEFT або RIGHT. У результаті виконуються внутрішнє ліве, зовнішнє ліве та зовнішнє праве з'єднання відповідно.

Приклад. Запит із трьома таблицями можна переписати у вигляді

```
SELECT t.nazva_tov, p2.nazva_post
FROM (tovary t JOIN postavky AS p1 ON t.kod_tov = p1.kod_tov)
```

```
JOIN postachalnyky AS p2 ON p1.kod_postac = p2.kod_postac
```

При цьому *використання дужок є обов'язковим*!

Зовнішні з'єднання

При виконанні внутрішніх з'єднань у результат запиту включаються лише ті записи таблиць, для яких є відповідні їм записи у інших таблицях. Якщо потрібно включати дані кортежів таблиці, для яких нема відповідників, використовуються ліві зовнішні об'єднання цієї таблиці із іншими таблицями.

Приклад. Вивести назви усіх товарів та дати їх поставок.

```
SELECT t.nazva_tov, p.data
FROM tovary as t LEFT OUTER JOIN postavky AS p ON
                                        t.kod_tov = p.kod_tov
```

Упорядкування кортежів результату запиту

Режим упорядкування задається у необов'язковій секції ORDER BY, яка має наступний синтаксис:

```
ORDER BY вираз1 [порядок1][, вираз2 [порядок 2] …]
```

На відмінну від вимог стандарту, у FoxPro у якості виразу можна використовувати лише імена стовпців, їх псевдоніми або порядкові номери у секції `SELECT`, а у Access — не можна використовувати синоніми.

Порядок сортування може приймати одне з двох допустимих значень `ASC` — порядок зростання (по замовчуванню) та `DESC` — порядок спадання.

```
SELECT *
FROM tovar
ORDER BY nazva_tov DESC, kod_tov

SELECT t.nazva_tov, p.*, t.cina*p.kilk AS vartist
FROM tovary AS t, postavky AS p
WHERE p.kod_tov = t.kod_tov
ORDER BY vartist
```

Задання числа рядків результату запиту

У секції `SELECT` перед переліком виразів можна вказувати додатковий параметр `TOP {n}` для того, щоб вказати, що із результату запиту потрібно відібрати лише перші *n* записів. У FoxPro параметр `TOP` можна використовувати лише у випадку наявності секції `ORDER BY`.

Приклад. Вивести коди трьох поставок із найбільшою кількістю одиниць товару

```
SELECT TOP 3 kod_postav
FROM tovary
ORDER BY kilk
```

Функції у SQL

Вбудовані функції поділяються на *функції одного рядка* та *агрегатні*, які оперують значеннями атрибутів множини кортежів. Загальний вигляд унарних агрегатних функцій:

```
ім'я_функції([ALL | DISTINCT] вираз).
```

Основними агрегатними функціями є COUNT, SUM, AVG, MIN, MAX.
Функція COUNT має дві фори.

Перша форма COUNT(*) повертає кількість рядків вхідної таблиці.

Друга форма COUNT(вираз) — кількість рядків таблиці, значення виразу для яких не рівне NULL. Для підрахунку кількості значень без повторів треба використовувати параметр DISTINCT (не підтримується у MS Access).

Приклади.

```
SELECT COUNT(*)
FROM postavky
WHERE YEAR(data) = 2013 AND MONTH(data) >= 6
```

```
SELECT COUNT(DISTINCT nazva_tov)
FROM tovary
```

```
SELECT SUM(p.kilk * t.cina)
FROM postavky AS p, tovary AS t
WHERE p.kod_tov = t.kod_tov
```

При обчисленні агрегатних функцій незаповнені поля (зі значенням NULL) ігноруються (крім функції COUNT(*)).

Групування записів

Групування дає змогу поділити множину кортежів таблиці на групи по ознаці співпадання значень одного або кількох атрибутів. Кожний вираз у секції SELECT *має приймати єдине значення для групи*, тобто він може бути або константою, або виразом ідентичним тому, що міститься у секції GROUP BY, або агрегатною функцією, яка оперує значеннями у межах групи.

```
SELECT nazva_tov, MIN(cina), MAX(cina)
FROM tovar
GROUP BY nazva_tov
```

Якщо у запиті присутні секції WHERE та GROUP BY, то рядки, які не задовольняють умову, вказану у WHERE, виключаються до виконання групування. Тому кількість рядків результату може бути меншою за кількість різних значень атрибута, вказаного у секції GROUP BY.

У секції WHERE *не можна вказувати агрегатні функції!*

Приклад. Вивести назви постачальників та кількість різних найменувань товарів, поставки яких вони здійснювали з 10.01 по 20.03.2014.

```
SELECT p2.kod_postac, COUNT(DISTINCT p1.kod_tov)
FROM postavky p1, postachalnyky p2
WHERE p1.kod_postac = p2.kod_postac AND data BETWEEN '2014-01-10'
AND '2014-03-20'
GROUP BY p1.kod_postac
ORDER BY 2
```

Можна виконувати групування по кільком стовпцям:

```
SELECT SUM(kilk), ROUND(AVG(kilk),1)
```

```
FROM postavky
GROUP BY kod_postac, kod_tov
```

Стандарт передбачає, що усі значення NULL утворюють одну групу.

Умови відбору групу. Секція HAVING

Групи, інформація на основі яких включається у результат запиту із групуванням, можуть відбиратися із використанням групових умов, які визначаються у секції HAVING.

Приклад. Вивести номери тих складів, на які було поставлено не менше 20 поставок, у кожній з яких було більше 10 одиниць товару.

```
SELECT sklad, COUNT(*)
FROM Postavky
WHERE kilk > 10
GROUP BY sklad
HAVING COUNT(*) >= 20
```

При *виконанні запитів*

1) спочатку виконується відбір рядків із використанням умови, вказаної у WHERE;
2) потім їх групування (GROUP BY);
3) насамкінець відбір груп на основі HAVING. У запитах із групуванням на вирази секції HAVING накладають ті самі вимоги, що й у секції SELECT (унікальність у межах групи).

Приклад. Вивести назви товарів, для яких різниця між мінімальною та максимальною цінами не менша за 10 грн.

```
SELECT nazva_tov
FROM tovary
GROUP BY nazva_tov
HAVING MAX(cina) <= MIN(cina) + 10
```

Вкладені запити

Вкладений запит — це розташований у круглих дужках запит, який присутній у секціях WHERE, HAVING, FROM запитів мови SQL.

Прості підзапити

Простий підзапит — це підзапит, результат якого *не залежить від зовнішнього запиту*. При виконанні запитів спочатку виконуються прості підзапити, а після цього — зовнішні запити.

Приклад. Вивести назву товару, який входив у поставку із кодом 11.

```
SELECT nazva_tov
FROM tovary
WHERE kod_tov = (SELECT kod_tov
         FROM postavky
         WHERE kod_postav = 11)
```

Приклад. Вивести назви товарів, ціна яких не менша за середню ціну усіх товарів.

```
SELECT nazva_tov
FROM tovary
WHERE cina >= (SELECT AVG(cina)
```

```
                FROM tovary)
```

Приклад. Вивести назву та ціну товарів, які входили до поставок найбільшого обсягу.

```
SELECT t.nazva_tov, t.cina
FROM tovary t, postavky p
WHERE p.kod_tov = t.kod_tov AND p.kilk = (SELECT MAX(kilk)
                                FROM postavky)
```

Якщо підзапит повертає більше ніж одне значення, виникає помилка.

Приклад. Для кожного товару вивести скільки разів він постачався.

```
SELECT kod_tov, (SELECT COUNT(*) FROM Postavky
WHERE Postavky.kod_tov = Tovary.kod_tov)
FROM Tovary
```

У FoxPro *не допускається*

- вказувати у підзапитах внутрішні підзапити;
- використовувати підзапити у секціях `SELECT, FROM та HAVING`;
- вказувати підзапит лівим аргументом у операторах порівняння.

Вкладені запити у операторі IN.

Підзапити можуть використовуватися у якості другого аргументу оператора `IN`. Це дає змогу уникнути обмеження єдиності результату підзапиту.

Приклад. Вивести адреси постачальників, які постачали товари на другий склад восени 2014 року.

```
SELECT Adresa
FROM Postachalnyky
WHERE kod_postac IN (SELECT kod_postac
                  FROM Postavky
                  WHERE sklad = 2 AND Data BETWEEN '2014-09-01' AND
'2014-11-30')
```

Корельовані підзапити

Результат корельованого підзапиту залежить від зовнішнього запиту. Підзапит є *корельованим*, якщо у його секціях `WHERE` та `HAVING` *наявні стовпці зовнішнього запиту*. Запити із корельованими підзапитами обробляються окремо для кожного рядка зовнішнього запиту.

Приклад. Вивести коди поставок та номери складів для тих поставок, розмір яких менший за середній розмір поставок для відповідного складу.

```
SELECT kod_postav, sklad
FROM postavky as p1
WHERE kilk < (SELECT AVG(kilk)
           FROM postavky AS p2
           WHERE p1.sklad = p2.sklad)
```

Приклад. Вивести назви постачальників, які виконали хоча би дві поставки на перший склад.

```
SELECT nazva_post
FROM Postachalnyky p1
WHERE 2 <= (SELECT COUNT(*)
        FROM Postavky p2
        WHERE sklad = 1 AND p1.kod_postac = p2.kod_postac)
```

Зображення

Зображення (views) — це *віртуальні таблиці*, у яких зберігаються результати запитів. Синтаксис:

```
CREATE VIEW ім'я_зображення AS
запит_SELECT
```

У FoxPro для створення зображень використовується команда `CREATE SQL VIEW`. Попередньо потрібно відкрити БД.

Приклад. Вивести назву постачальників, для яких середня кількістю одиниць товару у поставці є найбільшою.

```
CREATE VIEW PostavkyAVG AS
SELECT kod_postac, AVG(kilk) AS sered_kilk
FROM Postavky
GROUP BY kod_postac
```

```
SELECT nazva_post
FROM Postachalnyky p1, PostavkyAVG p2
WHERE p1.kod_postac = p2.kod_postac AND p2.sered_kilk =
     SELECT MAX(sered_kilk)
     FROM PostavkyAVG)
```

Для видалення непотрібних зображень використовується команда

```
DROP VIEW ім'я_зображення
```

Наприклад,

```
DROP VIEW SkladView
```

Добавлення записів

Для добавлення нових кортежів (записів) у таблицю використовується інструкція `INSERT INTO`. Синтаксис:

```
INSERT INTO ім'я_таблиці [(ім'я_стовпця [,ім'я_стовпця]...)]
VALUES (значення [, значення]...)
```

Приклад.

```
INSERT INTO Tovary2 (kod_tov, nazva_tov) VALUES (103, 'печиво')
```

Використання запитів при добавленні даних

З використанням запитів можна добавляти до таблиці *множину рядків*.

```
INSERT INTO ім'я_таблиці [(ім'я_стовпця [,ім'я_стовпця]...)]
запит
```

Приклад. Додати у таблицю Tovary2 записи про воду із таблиці Tovary

```
INSERT INTO Tovary2 (kod_tov, nazva, cina)
SELECT kod_tov, nazva_tov, cina
FROM Tovary
WHERE LCASE(Tovary.nazva_tov) LIKE 'сир'
```

Видалення записів

Для видалення записів використовується інструкція DELETE. Синтаксис:

```
DELETE FROM Таблиця [[AS] псевдонім]
[WHERE умова];
```

Якщо секція WHERE відсутня, то видаляються усі рядки.

Приклад. Видалити інформацію про товари, які жодного разу на протязі останнього року.

```
DELETE FROM Tovary
WHERE NOT kod_tov IN (SELECT kod_tov
                      FROM Postavky
                      WHERE DATE() - data <= 365)
```

Приклад. Видалити інформацію про постачальників, для яких сумарна кількість поставлених одиниць товарів найменша.

```
CREATE VIEW SumPost AS
     SELECT kod_postac, sum(kilk) AS sum_kilk
     FROM Postavky
     GROUP BY kod_postac
```

```
DELETE FROM Postachalnyky
WHERE kod_postac IN (SELECT kod_postac
                     FROM SumPost
                     WHERE sum_kilk = (SELECT MIN(sum_kilk)
                                       FROM SumPost))
```

Модифікація даних

Для модифікації існуючих даних використовується інструкція UPDATE.

```
UPDATE Таблиця [[AS] псевдонім]
SET column1 = exp1 [, column2 = exp2...]
[WHERE condition]
```

Якщо секція WHERE відсутня, то модифікуються усі рядки.

Приклад. Зменшити на 20% ціну найдорожчого товару.

```
UPDATE tovary
SET cina = cina*0.8
WHERE cina = (SELECT MAX(cina) FROM tovary)
```

Приклад варіанта завдань модульного контролю

БД "Продажі автомобілів" містить таблиці "Автомобілі", "Клієнти" та "Продажі". У таблиці "Автомобілі" зберігається код, марка, рік випуску, колір та ціна автомобілів. У таблиці "Клієнти" — код, прізвище, адреса та відсоток знижки для відповідних осіб. У таблиці "Продажі" — код продажі, дата, код клієнта та код автомобіля. Виконати наступні завдання:

1. Зменшити на 10% ціну автомобілів жовтого кольору, випущених понад 5 років тому.
2. Видалити записи про автомобілі, ціна яких не менша за середню ціну автомобілів виробника "Opel".
3. Вивести прізвища двох клієнтів, які придбали найбільшу кількість автомобілів у 2016 році.
4. Видалити записи про автомобілі, які жодного разу не придбали у поточному році.
5. Клієнтам, які придбали не менше трьох автомобілів, збільшити відсоток знижки на 5.
6. Вивести марки автомобілів, для яких сумарна ціна продаж клієнтам міста Київ (з урахуванням їх знижок) була найбільшою.
7. Створити таблицю "Звичайні клієнти", яка має таку саму структуру, що й таблиця клієнти та записати у нею інформацію про тих клієнтів, які придбали рівно один новий автомобіль.

Рекомендована література

1. Завадський І. О. Основи баз даних. — К.:, 2011. — 192 с.
2. Когаловский М. Р. Энциклопедия технологий баз данных. — М.: Финансы и статистика, 2002. — 800 с.
3. Кузнецов С. Д. Основы баз данных. — 2-е изд. — М.: БИНОМ, 2007. — 484 с.
4. Бураков П. В., Петров В. Ю. Введение в системы базы данных. — СПб., 2010. — 129 с.
5. Дейт К. Дж. Введение в системы баз данных. — 8-е изд. — М.: Вильямс, 2005. — 1328 с.
6. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. — 3-е изд. — М.: Вильямс, 2003. — 1436 с.
7. Гарсиа-Молина Г., Ульман Дж., Уидом Дж. Системы баз данных. Полный курс. — Вильямс, 2003. — 1088 с.