Уважаемые студенты групп!

Вашему вниманию представлена лекция на тему «Основы БД в Delphi (Продолжение лекции)
Основы языка SQL». Лекция рассчитана на 6 часов

Задание

1. Прочитать внимательно лекцию.
2. Законспектировать лекцию в рабочую тетрадь не менее 3-6 страницы рукописного текста. В конспекте лекции обязательно должно быть приведены примеры.
3. Ответить письменно в рабочей тетради на контрольные вопросы.
4. Дата предоставления полного фотоотчета до 11.04.2023

С уважением Ганзенко Ирина Владимировна
!!! Если возникнут вопросы обращаться по телефону 0721134803 (вацап), +79591134803(телеграмм)
disobuch.ganzenko2020@mail.ru

Основы БД в Delphi (Продолжение лекции)

Основы языка SQL

План

1 Общие сведения

Язык SQL (Structured Query Language – язык структурированных запросов) был создан Microsoft в конце 70-ых годов и получил через некоторое время широкое распространение. Он позволяет формировать весьма сложные запросы к базам данных. Запрос – это вопрос к базе данных, возвращающий запись или множество записей, удовлетворяющих вопросу.

К сожалению, SQL а настоящее время недостаточно стандартизован. Существует стандарт SQL ANSI, но существует и множество диалектов, с которыми работают различные системы. Например, Sybase SQL Server и Microsoft SQL используют синтаксис, существенно отличающийся от стандарта ANSI. InterBase, Oracle и многие другие серверы в основном придерживаются стандарта ANSI, но каждый разработчик вносит в него и свои усовершенствования. Ниже изложение будет основываться на диалекте, принятом в локальном сервере InterBase, с которым вы познакомитесь в главе

5. Впрочем, поскольку речь будет идти только об основных операторах языка, расхождение в их синтаксисе между различными диалектами невелико. А чтобы действительно изучить SQL, надо обратиться к документации той системы, с которой вы работаете.

Delphi позволяет приложению при помощи запросов SQL использовать данные:

- Таблиц PARADOX и dBase – используется синтаксис локального SQL
- Локального сервера InterBase – полностью поддерживается соответствующий синтаксис
- Удаленных серверов SQL через драйверы SQL Links

Общие правила синтаксиса SQL очень просты. Язык SQL не чувствителен к регистру, так что, например, рассмотренный ниже оператор Select можно писать и SELECT, и Select, и select. Если используется программа из нескольких операторов SQL, то в конце каждого оператора ставится точка с запятой «;». Впрочем, если вы используете всего один оператор, то точка с запятой в конце не обязательна. Комментарий может писаться в стиле * Си: /*<комментарий>*/, а в некоторых системах и в стиле Pascal: {<комментарий>}. Вот, собственно, и все правила.

В примерах последующих разделов этой главы и на протяжении всей книги будет в основном использоваться база данных InterBase, описанная в разделе 1.5 и содержащая сведения о сотрудниках некоторой организации. Несложные опреции по созданию этой базы данных описаны в Приложении 1. База данных имеет псевдоним ib и зарегистрирована на пользователя «А» с паролем «1». В некоторых случаях будет использоваться аналогичная база данных Paradox с псевдонимом dbP. Ее создание описано в Приложении 2.

2.2 Оператор выбора Select

2.2.1 Отбор записей из таблицы

В этом разделе мы познакомимся с наиболее часто используемым оператором SQL – оператором выбора Select. Этот оператор возвращает одно или множество значений, которые могут представлять собой значения указанных полей записей, удовлетворяющих заданному условию и упорядоченных по заданному критерию.

Хотя мы еще не рассматривали компонент Delphi Query, предназначенный для работы с SQL, но при знакомстве с оператором Select полезно сразу пробовать записывать его в различных вариантах и с помощью компонента Query смотреть получающиеся результаты.

Поэтому откройте новое приложение Delphi, перенесите на форму компонент Query со страницы библиотеки Data Access и установите его свойство DatabaseName равным ib (или равным dbP при использовании базы данных Paradox). Поместите на форму компонент DataSource и в его свойстве DataSet задайтеQuery 1. Поместите также на форму компонент DBGrid и в его свойстве DataSource задайте DataSourcel.

Теперь ваше тестовое приложение для экспериментов с языком SQL готово. Операторы SQL вы можете писать в свойстве SQL компонента Queryl, а чтобы увидеть результаты выполнения написанного оператора, вам надо будет устанавливать значение свойства Active компонента Queryl в true. Это надо будет делать после записи каждого нового оператора.

Теперь начнем рассмотрение оператора Select. Одна из форм этого оператора имеет синтаксис:

SELECT <список имен полей> FROM <таблица>

WHERE <условие отбора> ORDER 3Y <список имен полей>;

Элементы оператора WHERE и ORDER BY не являются обязательными.

Элемент WHERE определяет условие отбора записей: отбираются только те, в которых условие выполняется. Элемент ORDER BY определяет упорядочивание возвращаемых записей.

<таблица> – это та таблица базы данных, из которой осуществляется отбор, например, Pers.

Начнем подробное рассмотрение данного оператора со списка долей после ключевого слова Select, содержащего имена тех полей таблицы, которые будут возвращены. Имена разделяются запятыми. Например, оператор

SELECT Fam, Nam, Par, Year__b FROM Pers

указывает, что следует вернуть поля Fam, Nam, Par и Year_b из таблицы Pers. Запишите его в свойстве SQL компонента Query 1, установите значение свойства Active компонента Queryl в true и посмотрите результаты.

Если указать вместо списка нолей символ «*» – это будет означать, что требуется вернуть все поля. Например, оператор

SELECT * FROM Pers

указывает, что следует вернуть поля Fam, Nam, Par и Year_b из таблицы Pers. Запишите его в свойстве SQL компонента Query 1, установите значение свойства Active компонента Queryl в true и посмотрите результаты.

Если указать вместо списка нолей символ «*» – это будет означать, что требуется вернуть все поля. Например, оператор

SELECT * FROM Pers

означает выбор всех полей.

В списке могут быть не только сами поля, но и любые выражения от них с арифметическими операциями +, –, *, /. После выражения может записываться псевдоним выражения в форме: AS <псевдоним>. В качестве псевдонима может фигурировать любой идентификатор, на который потом можно будет при необходимости ссылаться. Указанный псевдоним будет при отображении результатов фигурировать в заголовке таблицы. Приведем пример использования выражения:

SELECT Fam, Nam, (2000-Year_b) AS Age FROM Pers

Этот оператор создает поле Age, вычисляемое по формуле (2000-Year_b).

При работе с некоторыми типами баз данных вы можете написать псевдоним по-русски. Например, если бы вы работали с базой данных dbP (база данных Paradox), то вы могли бы изменить предыдущий оператор следующим образом:

SELECT Fam AS Фамилия, Nam AS Имя,

(2000-Year_b) AS Возраст FROM Pers

В этом случае вы увидели бы, что в заголовках таблицы появились осмысленные русские заголовки:

Фамилия	Имя	Возраст
Иванов	Иван	50
Петров	Петр	40
…	…	…

При работе с базами данных InterBase использование русских псевдонимов невозможно. Впрочем, позднее, при рассмотрении компонента Query вы увидите, что эффекта получения русских заголовков можно добиться и другими способами, не вводя русских обозначений в оператор Select.

В выражениях для полей могут использоваться строковые константы и операция «||», означающая сцепление строк. Например, оператор

```
SELECT "Отдел: " || Dep, "Фамилия: " || Fam FROM Pers
```

будет выдавать строки вида:

Отдел: Бухгалтерия. Фамилия: Иванов

Если желательно в подобном виде выдавать значения полей, содержащих не символы, а, например, числа, то можно воспользоваться функцией Cast, приводящей поле к указанному в ней типу. Например, в операторе

```
SELECT "Отдел: " || Dep, "Фамилия: " || Fam,
                "г.р.: " И Cast(Year_b AS CHAR{4)) FROM Pers
```

функция Cast преобразует значение числового поля Year_b в строку из четырех символов. Результат будет выдаваться строками вида:

Отдел: Бухгалтерия Фамилия: Иванов г.р.: 1950

Теперь рассмотрим форму представления условия отбора, задаваемого после ключевого слова WHERE. Это условие определяет критерий, по которому отбираются записи. Оператор Select отбирает только те записи, в которых заданное условие истинно. Условие может включать имена полей (кроме вычисляемых полей), константы, логические выражения, содержащие арифметические операции, логические операции and, or, not и операции отношения:

= равно

> больше

>= больше или равно

< меньше

<= меньше или равно

!= или <> не равно

Like наличие заданной последовательности символов

between ... and диапазон значений

in соответствие элементу множества

Первые шесть операций очевидны. Например, оператор

SELECT Fam FROM Pers WHERE Sex='ж' and Year_b > 1960

отберет записи, относящиеся к женщинам, родившимся после 1960 года.

Операция Like имеет синтаксис:

<поле> LIKE '<последовательность символов>'

Эта операция применима к полям типа строк и возвращает true, если в строке встретился фрагмент, заданный в операции как <последовательность символов>. Заданным символам может предшествовать и их может завершать символ процента «%», который означает – любое количество любых символов. Если символ процента не указан, то заданная последовательность символов должна соответствовать только целому слову. Например, условие

Fam LIKE 'A%'

означает, что будут отобраны все записи, начинающиеся с заглавной русской буквы «А» (операция Like различает строчные и прописные символы). Условию

Fam LIKE 'Иванов%'

будут удовлетворять фамилии «Иванов» и «Иванова», а условию

Fam LIKE '%ван%'

кроме этих фамилий будет удовлетворять, например, фамилия «Иванников».
Операция between ... and имеет синтаксис:

<поле> between <значение> and <значение>

и задает для указанного поля диапазон отбираемых значений. Например, оператор

```
SELECT Fam, Year_b FROM Pers
        WHERE Year_b BETWEEN 1960 AND 1970
```

отберет записи сотрудников в заданном диапазоне возраста.

Операция In имеет синтаксис:

```
<поле> in (<множество>)
```

и отбирает записи, в которых значение указанного поля является одним из элементов указанного множества. Например, оператор

```
SELECT Fam, Year_b FROM Pers
        WHERE Fam IN('Иванов','Петров','Сидоров')
```

отберет записи сотрудников с заданными фамилиями, а оператор

```
SELECT Fam, Year_b FROM pers WHERE Year_b IN(1950,1960)
```

отберет записи сотрудников указанных годов рождения.

Элемент оператора Select, начинающийся с ключевых слов ORDER BY, определяет упорядочивание (сортировку) записей. После этих ключевых слов следует список полей, определяющих сортировку. Можно указывать только поля, фигурирующие в списке отобранных (в списке после ключевого слова SELECT).

Если в списке сортировки указано только одно поле, то сортировка производится по умолчанию в порядке нарастания значений этого поля. Например, оператор

```
SELECT Dep, Fam, Year_b FROM Pers ORDER BY Year_b
```

задает упорядочивание возвращаемых значений по нарастанию года рождения. Если желательно располагать результаты по убыванию значений, то после имениполя добавляется ключевое слово DESC:

```
SELECT Dep, Fam, Year_b FROM Pers ORDER BY Year_b DESC
```

Если в списке после ORDER BY перечисляется несколько полей, то первое из них – главное и сортировка проводится прежде всего по значениям этого поля.

Записи, имеющие одинаковое значение первого поля упорядочиваются по значениям второго поля и т.д. Например, оператор

```
SELECT Dep, Fam, Year_b FROM Pers ORDER BY Dep, Fam
```

сортирует записи прежде всего по отделам (значениям поля Dep), а внутри каждого отдела – по алфавиту. Оператор

```
SELECT Dep, Fam, Year_b, Sex
    FROM Pers ORDER BY Dep, Sex, Fam
```

сортирует записи по отделам, полу и алфавиту.

После ключевого слова Select в оператор могут вставляться ключевые слова DISTINCT или ALL. Первое из них означает, что в результирующий набор данных не включаются повторяющиеся записи. Повторяющимися считаются те записи, в которых совпадают значения полей, перечисленных в списке оператора Select. Ключевое слово ALL означает включение всех записей. Оно подразумевается по умолчанию, так что вставлять его в оператор не имеет смысла. Приведем пример использования DISTINCT. Оператор

```
SELECT DISTINCT Dep FROM Pers
```

выдаст список подразделений, в которых работают сотрудники.

2.2.2 Совокупные характеристики

Оператор Select позволяет возвращать не только множество значений полей, но и некоторые совокупные (агрегированные) характеристики, подсчитанные по всем или по указанным записям таблицы. Одна из функций, возвращающих такие совокупные характеристики, count(<условие>) – количество записей в таблице, удовлетворяющих заданным условиям. Например, оператор

```
SELECT count( * ) FROM Pers
```

подсчитает полное количество записей в таблице Pers. А оператор

```
SELECT count{*)FROM Pers WHERE Dep="Цех 1"
```

выдаст число записей сотрудников цеха 1.

Оператор, использующий ключевое слово DISTINCT (уникальный), выдаст число неповторяющихся значений в указанном поле. Например, оператор

```
SELECT count(DISTINCT Dep) FROM
```

Pers вернет число различных подразделений, упомянутых в поле Dep таблицы Pers.

Функции min(<поле>), max(<поле>), avg(<поле>), sum(<поле>) возвращают соответственно минимальное, максимальное, среднее и суммарное значения указанного поля. Например, оператор

```
SELECT min(Year_b), max (Year_b) , avg (Year__b) FROM Pers
```

вернет минимальное, максимальное и среднее значение года рождения, а оператор

```
SELECT min(2000-Year_b), max(2000-Year_b),
    avg(2000-Year_b) FROM Pers WHERE Dep='Бухгалтерия'
```

выдаст вам аналогичные данные, но относящиеся к возрасту сотрудников бухгалтерии.

В операторе Select вы можете указывать не только суммарные характеристики, но и любые выражения от них. Например, оператор

```
SELECT 2000-(min(Year_b)+max{Year_b))/2 FROM Pers
    WHERE Dep='Бухгалтерия'
```

выдаст моду (среднее между максимальным и минимальным значениями) возраста сотрудников бухгалтерии.

При использовании суммарных характеристик надо учитывать, что в списке возвращаемых значений после ключевого слова SELECT могут фигурировать или поля (в том числе вычисляемой второй таблице). Возможны и другие виды объединений, которые выдают записи независимо от того, есть ли соответствующее поле во второй таблице. Это внешние объединения (outer join). Их три типа: левое, правое и полное. Левое объединение (обозначается ключевыми словами LEFT OUTERJOIN ... ON) включает в результат все записи первой таблицы, даже те, для которых не имеется соответствия во второй. Правое объединение (обозначается ключевыми словами RIGHT OUTER JOIN ... ON) включает в результат все записи второй таблицы, даже если, им нет соответствия в записях первой. Полное объединение (обозначается ключевыми словами FULL OUTER JOIN ... ON) включает в результат объединение записей обеих таблиц, независимо от их соответствия.

Пусть, например, у вас есть таблица сотрудников некоей ком-пании Pers и есть таблица Chef, в которой занесены данные на членов совета директоров этой компании. В число членов совета входят и сотрудники компании, и посторонние лица. Для определенности положим, что в таблице Pers имеются записи на сотрудников «Иванов» и «Петров», причем Петров является членом совета, а Иванов – нет. В таблице Chef имеются записи на членов совета «Петров» и «Сидоров», причем Сидоров – не сотрудник компании. Тогда оператор

SELECT * FROM Pers LEFT OUTER JOIN Chef

ON Pers.Fam = Chef.Far

выдаст результат вида:

Поля таблицы Pers		Поля таблицы Chef	
Иванов	……		……
Петров	……	Петров	……

Оператор задал левое объединение таблицы Pers (она указана после ключевого слова FROM) с таблицей Chef (она указана после ключевых слов LEFTOUTER JOIN). Условие объединения указано после ключевого слова ON и заключается в совпадении фамилий.

Как показано, результат включает все поля и таблицы Pers, и таблицы Chef. Число строк соответствует числу записей таблицы Pers. В строках, относящихся к записям, для которых в Chef не нашлось соответствие, поля таблицы Chef остаются пустые.

Оператор правого объединения

SELECT * FROM Pers RIGHT OUTER JOIN Chef

ON Pers.Fam = Chef.fam

выдаст результат вида:

Поля таблицы Pers		Поля таблицы Chef	
Петров	……	Петров	……
		Сидоров	……

Число строк соответствует числу записей таблицы Chef. В строках, относящихся к записям, для которых в Pers не нашлось соответствие, поля таблицы Pers остаются пустые.

Оператор полного объединения

SELECT * FROM Pers FULL OUTER JOIN Chef ON Pers.Fam = Chef.Fa
m

выдаст результат вида:

Поля таблицы Pers		Поля таблицы Chef	
Иванов	……		
Петров	……	Петров	……
		Сидоров	……

В нем к строкам, относящимся к таблице Pers, добавлены строки, относящиеся к таблице Chef, для которых не нашлось соответствия в таблице Pers.

2.3 Операции с записями

В этом и нескольких последующих разделах вы уже не сможете проверять операторы SQL с помощью вашего тестового приложения. Или обойдитесь пока без проверки, или посмотрите сначала раздел 5.3 и проверяйте операторы с помощью программы WISQL.

Вставка новой записи в таблицу осуществляется оператором Insert, который может иметь вид:

INSERT INTO <имя таблицы> (<список полей>)

VALUES (<список значений>)

В списке перечисляются только те поля, значения которых известны. Остальные могут опускаться. Для пропущенных полей значения берутся по умолчанию (если значения по умолчанию заданы) или поля остаются пустыми.

Например:

```
INSERT INTO Pers (Num,Fam, Nam, Par, Sex)

    VALUES (12,'Иванов', 'Андрей', 'Андреевич', 'м')
```

При создании таблицы на значения поля могут накладываться ограничения. Например, оператор

```
CREATE TABLE Pers(
    Num smallint Not Null Primary Key,
    Fam char(20) Not Null,
    Nam char(20) Not Null,
    Par char(20) Not Null,
    Year_b smallint DEFAULT 1950
                CHECK( (Year_b>1917)and(Yeaj:_b<1980) ));
```

устанавливает ограничение на год рождения: больше 1917 и меньше 1980. Ограничения записываются после ключевого слова CHECK.

Создаваемую таблицу можно связать как дочернюю с одной из уже созданных таблиц. Синтаксис такой связи следующий:

```
FOREIGN KEY (<список ключей – полей данной таблицы>)
REFERENCES  <имя родительской таблицы>
            <список ключевых полей родительской таблицы>
ON DELETE   <действие> ON UPDATE   <действие>
```

В этом определении после слов FOREIGN KEY записывается список полей таблицы, являющихся ключами, по которым данная таблица связывается с головной (родительской). Список соответствующих ключевых полей родительской таблицы после ключевого слова REFERENCES может опускаться, если ключом является первичный ключ родительской таблицы.

Не обязательные элементы определения ON DELETE и ON UPDATE указывают, какие действия должны выполняться, если запись родительской таблицы удаляется или модифицируется. Возможные значения действий:

NO ACTION при наличии подчиненных записей в дочерней таблице удаление или изменение соответствующей записи головной таблицы запрещено

CASCADE при удалении записи в головной таблице происходит удаление всех подчиненных ей записей в дочерней таблице; при изменении записи в головной таблице изменяется значение ключевого поля во всех подчиненных ей записях дочерней таблицы

SET DEFAULT при удалении или изменении записи головной таблицы ключевому полю во всех подчиненных записях дочерней таблицы присваивается значение по умолчанию

SET NULL в ключевое поле подчиненных записей дочерней таблицы заносится NULL

В нашей демонстрационной базе данных головной таблицей, очевидно, должна быть таблица описания подразделений Dep с полем Dep – подразделение, являющимся ее первичным ключом. Дочернюю таблицу Pers можно связать с ней по ее полю Dep – подразделению, в котором работает сотрудник. При переименовании подразделения (изменении записи в головной таблице) в дочерней таблице тоже должно каскадно изменяться название этого подразделения во всех содержащих его записях. Для осуществления этого надо задать режим ON UPDATE CASCADE. При ликвидации подразделения (удалении записи в головной таблице) возможно, по-видимому, несколько вариантов. Если указать режим ON DELETE CASCADE, то все записи в таблице Pers, относящиеся к этому подразделению, автоматически удалятся, т.е. все сотрудники подразделения окажутся уволенными и сведения о них исчезнут. Вероятно, это неудачный вариант, так как в реальной жизни сотрудники ликвидированного подразделения частично могут быть переведены в другие подразделения. Логичнее указать режим ONDELETE SET NULL. В этом случае записи сотрудников ликвидированного подразделения будут помечены значением NULL в поле Dep. И в дальнейшем администрация может решать судьбу каждого такого сотрудника индивидуально. Но значение NULL отсутствует в головной таблице и, значит, соответствующие записи таблицы Pers окажутся не привязанными к таблице Dep. Вероятно, наиболее удобный вариант – установить режим ON DELETE SET DEFAULT, в таблице Pers для поля Dep ввести значение по умолчанию (например, «Неизвестный») и в головной таблице Dep ввести запись с аналогичным значением поля Dep. Тогда запросы, создающие эти таблицы, могут иметь вид:

```
/* Создание таблицы Dep */
create table Dep(
        Dep char(15)  Not Null Primary Key,
        Proisv char(l)
  );

/* Создание таблицы Pers */
create table Pers(
        Num smallint Not Null Primary Key,
        Dep char(15) DEFAULT "Неизвестный",
        Fam char(20) Not Null,
        Nam char(20) Not Null,
```

```
        Par char(20) Not Null,
        …
    FOREIGN KEY (Dep) REFERENCES Dep
    ON DELETE SET DEFAULT ON UPDATE CASCADE
    );
```

При создании нескольких таблиц можно обеспечивать целостность данных на уровне ссылок. Это означает, что в одной из таблиц некоторое поле может иметь только такие значения, которые содержатся в некотором поле другой таблицы. В нашем примере, очевидно, при создании таблицы Pers желательно обеспечить, чтобы в значение ее поля Dep невозможно было ввести подразделение, отсутствующее в таблице Dep. Это можно сделать следующими операторами:

```
/* Создание таблицы Dep */ create table Dep(
Dep char(15)  Not Null Primary Key,
Proisv char(1)
);

/* Создание таблицы Pers */
create table Pers(
  Num smallint Not Null Primary Key,
   Dep char{15) DEFAULT "Неизвестный",
   Fam char(20) Not Null,
   Nam char(20) Not Null,
   Par char(20) Not Null,
…
FOREIGN KEY (Dep) REFERENCES Dep  -
ON DELETE SET DEFAULT ON UPDATE CASCADE,
CONSTRAINT Dep CHECK{EXISTS(
      SELECT Dep FROM Dep WHERE Pers.Dep = Dep.Dep))

) ;
```

После ключевого слова CONSTRAINT следует имя поля Dep данной таблицы, а затем записано условие проверки допустимости значения данного поля. В приведенном примере это условие сводится к тому, что значение поля Dep таблицы Pers должно встречаться среди значений поля Dep таблицы Dep.

Мы рассмотрели основные формы оператора создания таблицы. Удаление таблицы осуществляется оператором Drop Table:

DROP TABLE <имя таблицы>

Надо учесть, что удаление таблицы в корне отличается от удаления в ней всех записей. При удалении даже всех записей сама таблица (ее структура) остается, а оператор Drop Table полностью уничтожает таблицу.

Модификация структуры существующей таблицы осуществляется оператором Alter Table:

ALTER TABLE <имя таблицы> <действие> <имя поля> <тип данных> .

В этом операторе <действие> может принимать значения ADD – добавить новое поле, или DROP – удалить существующее поле. Если поле добавляется, то для него надо указывать <тип данных>. Если поле удаляется, то тип данных не указывается. Приведем пример оператора модификации структуры:

```
ALTER TABLE Pers DROP Year__b, ADD Age integer
```

2.5 Операции с индексами

Индексы существенно ускоряют процесс поиска и упорядочивания записей таблицы. Если в операторе Select содержится элемент упорядочивания ORDER BY, то если перечисляемые поля совпадают с определенными в индексе – упорядочивание будет использовать этот индекс и произойдет с малыми затратами времени. В противном случае индекс использоваться не будет и упорядочивание потребует большего времени. Индекс будет использоваться и в том случае, если в ORDER BYперечислены не все поля индекса, а какие-то первые из них. Например, если создан индекс по полям p1, p2, p3, то при выполнении запроса

```
Select … ORDER BY pl,p2
```

индекс будет использоваться, а запрос

```
Select … ORDER BY pl,p3
```

будет обрабатываться без использования индекса.
Создание нового индекса осуществляется оператором Create Index:

```
CREATE INDEX <имя индекса> ON <имя таблицы >
                                        <список полей>
```

Например:

```
CREATE INDEX depyear ON Pers Dep, Year_b
```

Удаление существующего индекса осуществляется оператором Drop Index:

```
DROP INDEX <имя таблицы >.<имя индекса>
```

Например:

```
DROP Index Pers . depyear
```

Если таблица многократно изменяется и в нее вносится много новых записей, индексы могут оказаться разбалансированы и их эффективность при выполнении запросов

уменьшается. В этом случае полезно проводить повторное создание и балансировку индекса последовательным применением операторов деактивации и активации:

ALTER INDEX <имя индекса> DEACTIVATE

ALTER INDEX <имя индекса> ACTIVATE

Перестройка индекса может производиться только в случае, если он в данный момент не используется в запросах.