Создание таблиц БД

Таблицы создаются командой **CREATE TABLE**.

Эта команда создает пустую таблицу, не содержащую записей. Очевидно, что данные в нее можно внести, например, с помощью команды **INSERT**.

В команде **CREATE TABLE** определяется имя таблицы, и набор имен полей. Кроме того, этой же командой оговариваются типы данных и длины полей.

Синтаксис команды **CREATE TABLE** следующий:

CREATE TABLE <имя таблицы>

(<имя поля1> <тип данных> [(<длина>)],

(<имя поля2> <тип данных> [(<длина>)]),

…).

Значение длины поля зависит от типа данных.

Если его не указывать, то СУБД сама назначает значение автоматически (для числовых данных такой вариант предпочтительнее).

Для данных типа ***CHAR*** указание размера обязательно. По умолчанию значение длины равно 1.

Пример:

```
CREATE TABLE STUDENTS
(NOM_ZACH INTEGER,
 SFAM CHAR (20),
 SNAME CHAR (10),
STIP DECIMAL)
```

Числовые типы

Точные числовые типы

К категории точных числовых типов в SQL относятся те типы, значения которых точно представляют числа. Типы данных этой категории распадаются на две части: целые типы (INTEGER и SMALLINT) и типы, допускающие наличие дробной части (NUMERIC и DECIMAL).

- **целочисленные**:
  tinyint 0-255,
  smallint *(от -32 768 до 32 767),*
  int *(от -2,147,483,648 до 2,147,483,647) и*
  bigint (от -2^63 до 2^63);

- **десятичные**: **decimal** и **numeric** (это - два названия одного и того же типа);

- **денежные**: **money** (от -2^63 до 2^63 - с точностью 4 знака после запятой) и **smallmoney** (от -214748.3648 до +214748.3647).;
- **с плавающей запятой**: **float** (от -1.79E + 308 до 1.79E + 308) и **real** (от -3.40E + 38 до 3.40E + 38).

с плавающей запятой:

float (от -1.79E + 308 до 1.79E + 308) и **real** (от -3.40E + 38 до 3.40E + 38

DECIMAL [(точность[,масштаб])]
Параметр **точность** указывает максимальное количество цифр вводимых данных этого типа (до и после десятичной точки в сумме), а параметр **масштаб** – максимальное количество цифр, расположенных после десятичной точки.

Строковые типы

В SQL Server предусмотрены две дублирующих разновидности полей для представления текстовых данных:

поля **Unicode** и не-**Unicode**.

Unicode - типы данных начинаются символом **n** (от слова national, то есть с поддержкой национальных символов).

Всего в SQL Server предусмотрены следующие типы для текстовых данных:

- **char/nchar** - строковые данные фиксированной длины;
- **varchar/nvarchar** - строковые данные переменной длины.

- При использовании типа **Char** значения длиной короче заданной дополняются пробелами до указанной длины. Максимальное значение длины – 8000 символов.
- При использовании типа **VarChar** значения длиной короче заданной не дополняются пробелами.

Если необходимо ввести значения большой длины можно использовать ключевое слово **max**, что позволяет определять столбцы до 2^{31} байтов.

varchar(max).

- **datetime** (8 байт, точность до 3,33 миллисекунд);
- **smalldatetime** (4 байта, точность до минуты).

В большинстве приложений вполне хватает **smalldatetime;**

Тип данных **UNIQUEIDENTIFIER** используется для хранения глобальных уникальных идентификационных номеров.

SQL_VARIANT -
Служит для хранения значений разных типов одновременно, таких как числовые значения, строки и даты.

Объявлять тип столбца как SQL_VARIANT следует только в том случае, если это действительно необходимо. Например, если столбец предназначается для хранения значений разных типов данных или если при создании таблицы тип данных, которые будут храниться в данном столбце, неизвестен.

Логический тип данных - хранит значения вида **true/false** (единица/ноль).

В SQL Server он представлен типом данных **boolean**.

- **GETDATE** ( ) –

Возвращает значение типа datetime, которое содержит дату и время компьютера, на котором запущен экземпляр SQL Server.

- **DATEDIFF** (datepart , startdate , enddate)– возвращает интервал времени, прошедшего между двумя временными отметками - *startdate* (начальная отметка) и *enddate* (конечная отметка). Этот интервал может быть измерен в разных единицах. Возможные варианты определяются аргументом *datepart*

DATEPART (*datepart* , *date*) — возвращает целое число, представляющее собой указанную аргументом *datepart* часть заданной вторым аргументом даты

Year - год	yy, yyyy
Quarter - квартал	qq, q
Month - месяц	mm, m
Dayofyear - день года	dy, y
Day - день	dd, d
Week - неделя	wk, ww

В ряде случаев функцию **DATEPART** можно заменить более простыми функциями.

DAY (*date*) - целочисленное представление дня указанной даты. Эта функция эквивалентна функции **DATEPART**(*dd, date*).

MONTH (*date*) - целочисленное представление месяца указанной даты. Эта функция эквивалентна функции **DATEPART**(*mm, date*).

YEAR (*date*) - целочисленное представление года указанной даты. Эта функция эквивалентна функции **DATEPART**(*yy, date*).

DATEDIFF(dd, time_out, time_in)

Пользовательские типы данных.

Могут использоваться при определении какого-либо специфического или часто употребляемого формата.

Создание пользовательского типа данных осуществляется выполнением системной процедуры:

sp_addtype
[@typename=]type,[@phystype=]
system_data_type
[,[@nulltype=]'null_type']

EXEC sp_addtype dt, DATETIME, 'NULL'

Удаление пользовательского типа данных происходит в результате выполнения процедуры sp_droptype type

Пример:

EXEC sp_droptype 'dt‘

http://www.intuit.ru/studies/courses/5/5/lecture/124?page=2

CREATE TYPE SSN
FROM varchar(10) NOT NULL ;

Преобразование типов

Для выполнения преобразований SQL Server содержит функции **CONVERT** и **CAST**, с помощью которых значения одного типа преобразовываются в значения другого типа, если такие изменения вообще возможны.

CONVERT и **CAST** могут быть взаимозаменяемыми.

CAST(выражение AS тип_данных)

CONVERT(тип_данных[(длина)], выражение)

Пример:

```
SELECT ‘сегодня ‘ +
CONVERT(VARCHAR(11),GETDATE())


CAST('1977.01.07‘ AS Datetime)
```

Оновные функции

– поиск подстроки

CHARINDEX (expressionToFind ,expressionToSearch[, start_location])

- вырезка

SUBSTRING (**expression ,start , length**)

- REPLACE

заменяет указанную подстроку первого операнда строкой, заданной в качестве второго операнда.

REPLACE(expression , string_pattern , string_replacement)

-REVERSE

- Возвращает строковое значение, где символы переставлены в обратном порядке справа налево.

- TRIM "отсекает" последовательности указанного символа в конце или начале заданной строки.

Временные таблицы

Временные таблицы похожи на обычные, однако они не предназначены для постоянного хранения данных. Они создаются, удаляются и используются как обычные таблицы.

Имена временных таблиц должны начинаться с символов **#** или **##**.

Временные таблицы удаляются при отключении пользователя от базы данных.

Временные таблицы используются так, как будто они входят в текущую базу данных, однако в действительности данные хранятся в **TEMPDB.**

В SQL Server существуют два типа временных таблиц: **локальные** и **глобальные**.

Локальные временные таблицы доступны лишь для своего владельца. Имена локальных временных таблиц начинаются с префикса #.

Глобальные временные таблицы доступны для всех пользователей, их имена должны начинаться с префикса **##**.

Создание ограничений

Декларативные ограничения при создании таблиц

При создании таблиц могут быть заданы декларативные ограничения целостности атрибутов:

- значения по умолчанию (**DEFAULT**),
- задание обязательности или необязательности значений (**NULL** или **NOT NULL**),
- условия проверки значения **(CHECK**),
- задание уникальность столбца **(UNIQUE)** .

Например, на значение стипендии может быть наложено ограничение (стипендия должна находиться в пределах от 500 до 750 тысяч рублей) по умолчанию значение стипендии равно 500 тыс. руб.

STIP MONEY DEFAULT 700 CHECK(STIP >=700 AND <=750)

Возраст сотрудника должен быть не менее 18 лет:

```
BIRTH_DAY DATE
  CHECK(DATEDIFF(YEAR,GETDATE(),BIRT
  H_DAY)>=18)
```

При создании ограничений необходимо учитывать следующее:

- ограничение, определенное для одного поля может ссылаться только на это поле и называется **ограничением на уровне поля**;

- Ограничение может ссылаться на несколько полей и называется **ограничением на уровне таблицы.**

ограничения **DEFAULT** должны быть ограничениями **на уровне поля**;

ограничения **CHECK** на уровне поля могут ссылаться только на одно поле;

ограничения **CHECK** на уровне таблицы могут ссылаться на любые поля таблицы;

ограничения не могут ссылаться на поля других таблиц.

Пример ограничения на уровне таблицы

```
CREATE TABLE  TestTable
 ( id int DEFAULT 1 NOT NULL,
   vcName varchar(50) NOT NULL,
   dBirthDate datetime,
   dDocDate datetime,
  CONSTRAINT CK_birthdate CHECK
   (dDocDate>dBirthDate AND <=getdate())
```

Ограничение целостности, включаемое в определение столбца, может быть эквивалентным образом выражено в виде табличного ограничения целостности.

Например, ограничение NOT NULL для поля **CFAM** может быть задано так:

CHECK (CFAM IS NOT NULL).

Часто для поля или группы полей требуется реализовать ограничение, связанное с уникальностью значений.

В этом случае в ограничение поля (группы) при создании таблицы помещают ключевое слово **UNIQUE**. Можно определить группу полей как уникальную, например, в таблице **USP** уникальными должны быть комбинации полей **NOM_ZACH** и **PKOD**:

UNIQUE (NOM_ZACH,PKOD)

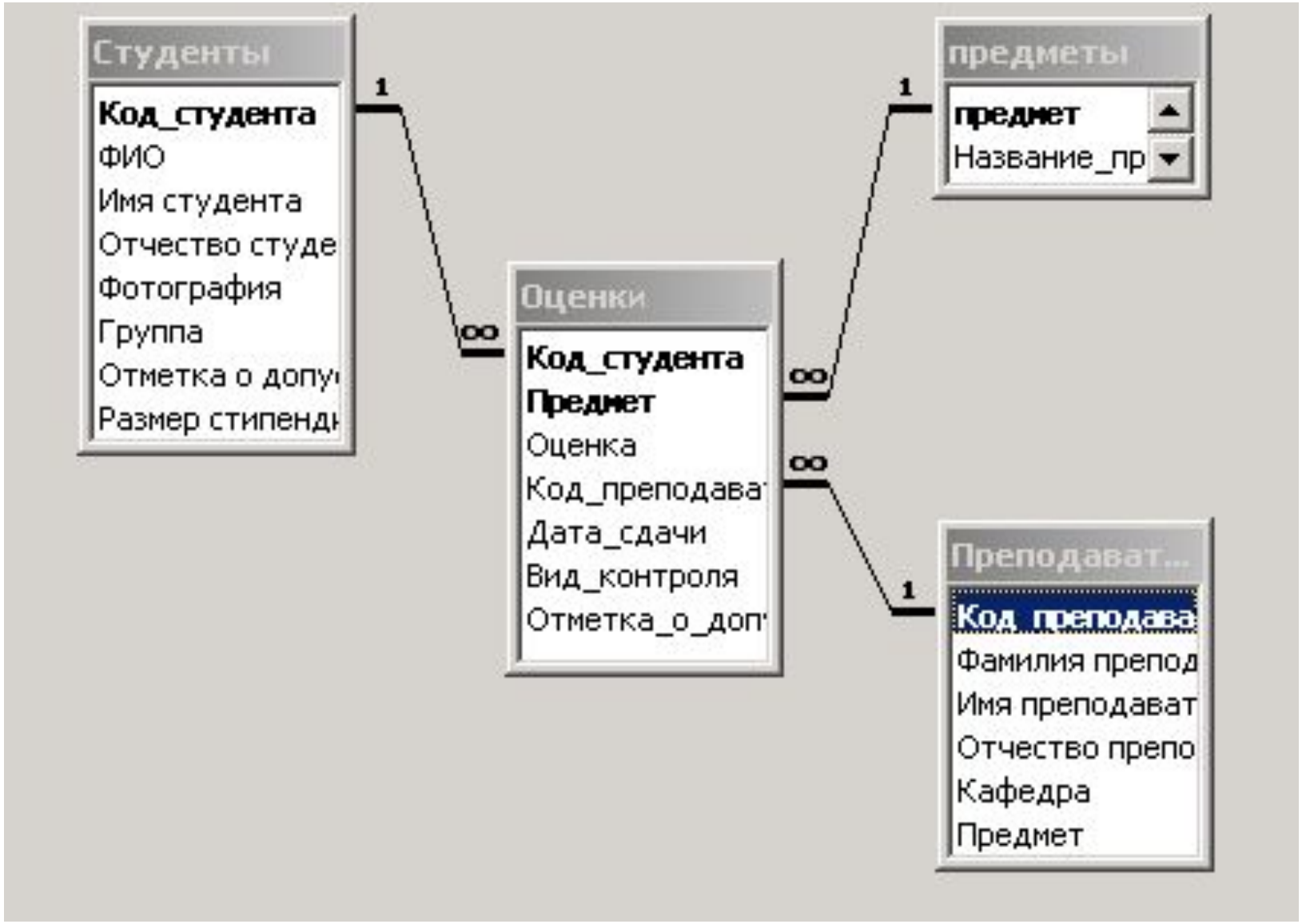
Ограничение **PRIMARY KEY** действует аналогично **UNIQUE**, но для таблицы должен быть определен только один первичный ключ, а уникальных полей может быть несколько.

Первичный ключ может быть составным (как в таблице **USP**, где ключ состоит из атрибутов **NOM_ZACH** и **PKOD**).

Для объявления составного первичного ключа требуется объявление на уровне таблицы.

PRIMARY KEY(NOM_ZACH, PKOD)

Ссылочная целостность


Студенты
Код_студента
ФИО
Имя студента
Отчество студе
Фотография
Группа
Отметка о допу
Размер стипенд
1
∞
Оценки
Код_студента
Предмет
Оценка
Код_преподава
Дата_сдачи
Вид_контроля
Отметка_о_доп
1
∞
предметы
предмет
Название_пр
1
∞
Преподават...
Код_преподава
Фамилия препод
Имя преподават
Отчество препо
Кафедра
Предмет

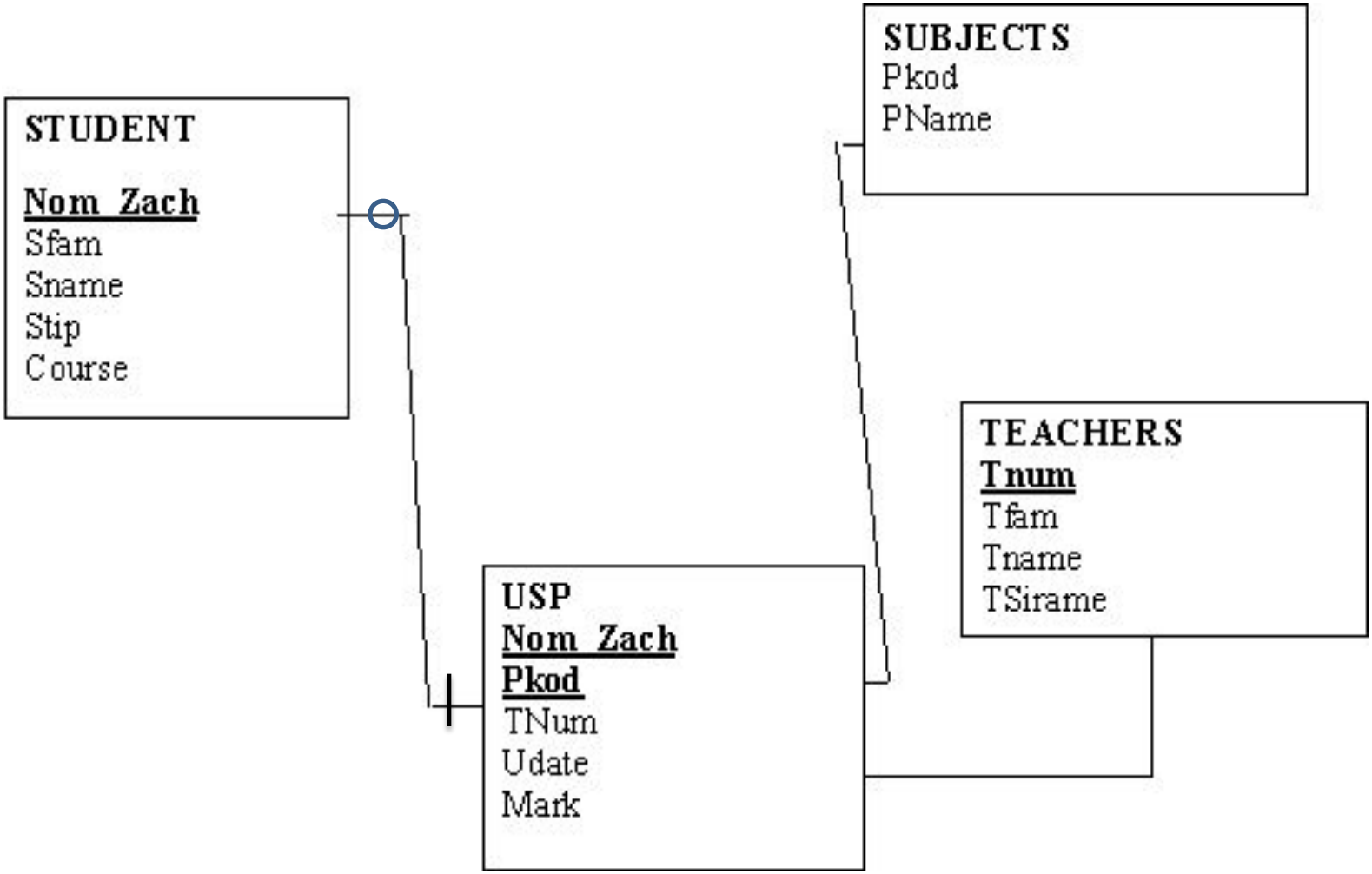

SUBJECTS
Pkod
PName
STUDENT
Nom_Zach
Sfam
Sname
Stip
Course
TEACHERS
Tnum
Tfam
Tname
TSirame
USP
Nom_Zach
Pkod
TNum
Udate
Mark

Таблица **USP** подчинена двум другим таблицам: **SUBJECTS** и **STUDENTS**. При этом таблица **USP** связана с таблицей **STUDENTS** обязательной связью.

Каждому значению атрибута **NOM_ZACH** в таблице **USP** **обязательно** должно соответствовать ровно одно значение этого же атрибута в таблице **STUDENTS.**

В таблице **USP** не может быть значений атрибута **NOM_ZACH**, которых нет в таблице **STUDENTS**. Связь с таблицей **SUBJECTS** также будет обязательной.

Для моделирования этих связей должны быть определены два внешних ключа (***FOREIGN KEY***) для полей **NOM_ZACH** и **PKOD**.

Для полей **NOM_ZACH** и **PKOD** должно быть задано значение **NOT NULL**, поскольку связь обязательная.

Ключ **FOREIGN KEY** ограничивает значения, которые можно ввести в БД так, чтобы заставить внешний и родительский ключи соответствовать принципу ссылочной целостности.

Синтаксис ограничения **FORIGN KEY:**

FOREIGN KEY <список полей> REFERENCES **<имя таблицы, содержащей родительский ключ>[список полей родительского ключа].**

Создадим таблицу **USP** с полем **NOM_ZACH**, и **PKOD** определенными в качестве внешних ключей:

```
CREATE TABLE USP
(NOM_ZACH INTEGER NOT NULL,
PKOD INTEGER  NOT NULL,
TNUM INTEGER,
UDATE DATE ,
MARK INTEGER,
PRIMARY KEY(NOM_ZACH, PKOD) ,

FOREIGN KEY (NOM_ZACH) REFERENCES  STUDENTS
   (NOM_ZACH),

FOREIGN KEY (PKOD) REFERENCES   SUBJECTS (PKOD))
```

Используя ограничения **FOREIGN KEY,** можно не указывать список полей родительского ключа, если родительский ключ имеет ограничение **PRIMARY KEY**.

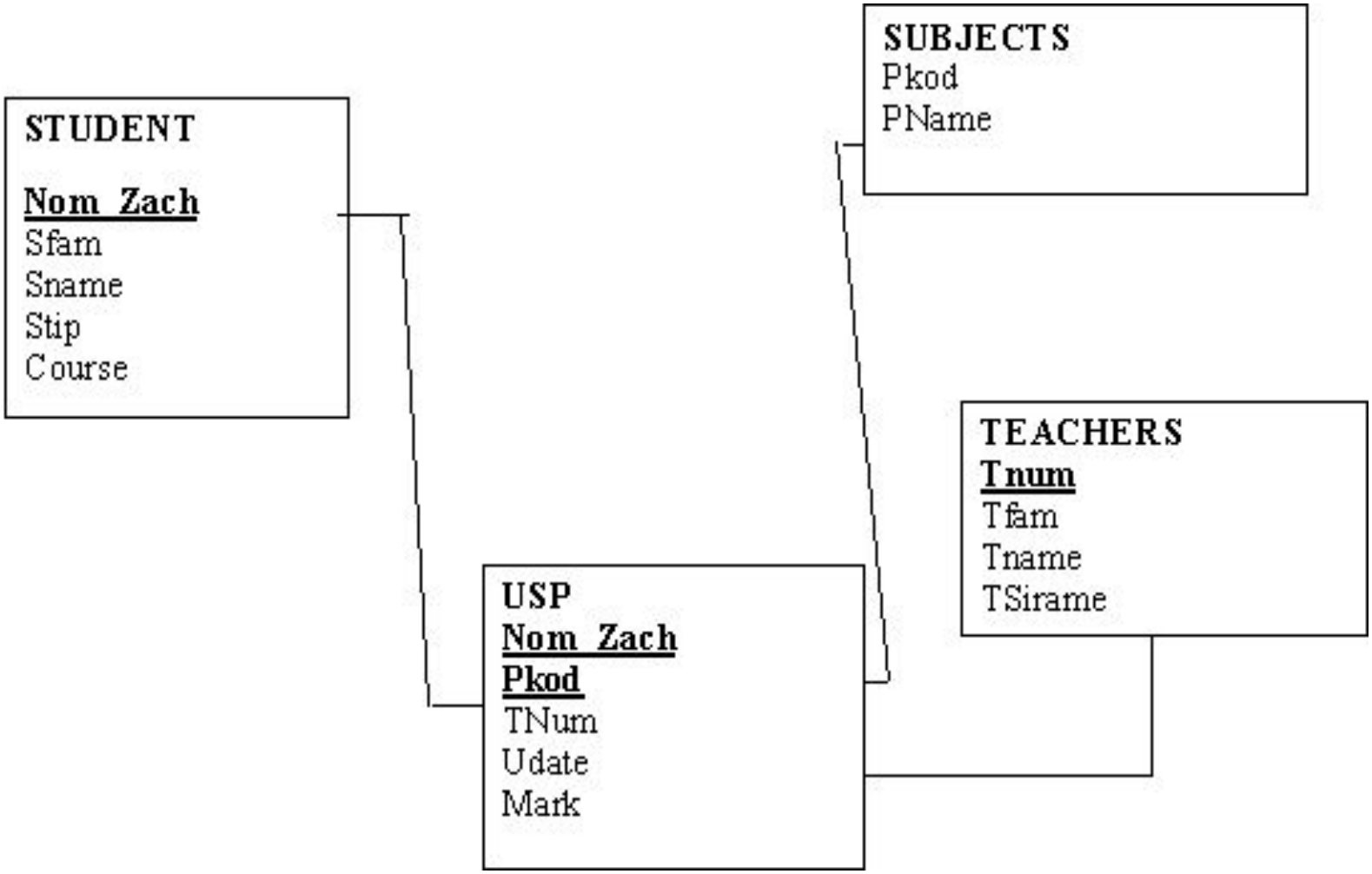
```
CREATE TABLE USP
(NOM_ZACH INTEGER NOT NULL FOREIGN KEY
    REFERENCES STUDENTS,
PKOD INTEGER NOT NULL FOREIGN KEY REFERENCES
    SUBJECT,
TNUM INTEGER,
UDATE DATE ,
MARK INTEGER,
PRIMARY KEY (NOM_ZACH,PKOD));
```

В случае употребления ключей со многими полями, обязательно выполнение условия, чтобы порядок полей во внешних и первичных ключах совпадал.

В соответствии со стандартом, изменение или удаление значений родительского ключа не допускается.

Это означает, что нельзя удалить данные о студенте из таблицы **STUDENTS** до тех пор, пока в таблице **USP** для него имеется какая-нибудь информация. Однако довольно часто возникают ситуации, когда необходимо удалить информацию о студенте, например, в случае его отчисления.

В таких случаях рассматривается возможность каскадирования или ограничения действий.


SUBJECTS
Pkod
PName
STUDENT
Nom_Zach
Sfam
Sname
Stip
Course
TEACHERS
Tnum
Tfam
Tname
TSirame
USP
Nom_Zach
Pkod
TNum
Udate
Mark

При необходимости изменить или удалить текущее ссылочное значение родительского ключа существует следующие возможности:

1. Запретить изменения (по умолчанию).
2. Сделав изменения в родительском ключе, произвести изменения во внешнем ключе автоматически (каскадное изменение).
3. Сделать изменение в родительском ключе и установить внешний ключ в **NULL** значение либо присвоить ему значение по умолчанию.

В пределах этих возможностей выполняются все команды модификации.

Итак, изменения в родительском ключе можно разделить на

- ограниченные (**NO ACTION**),
- каскадируемые (**CASCADE**),
- пустые (**SET NULL**) ,
- устанавливающие значения по умолчанию (**SET DEFAULT**).

Предположим, что есть необходимость в изменении номера зачетной книжки, причем оценки должны сохраниться у этого же студента с новым номером. В этом случае следует указать команду **UPDATE** с каскадируемыми изменениями.

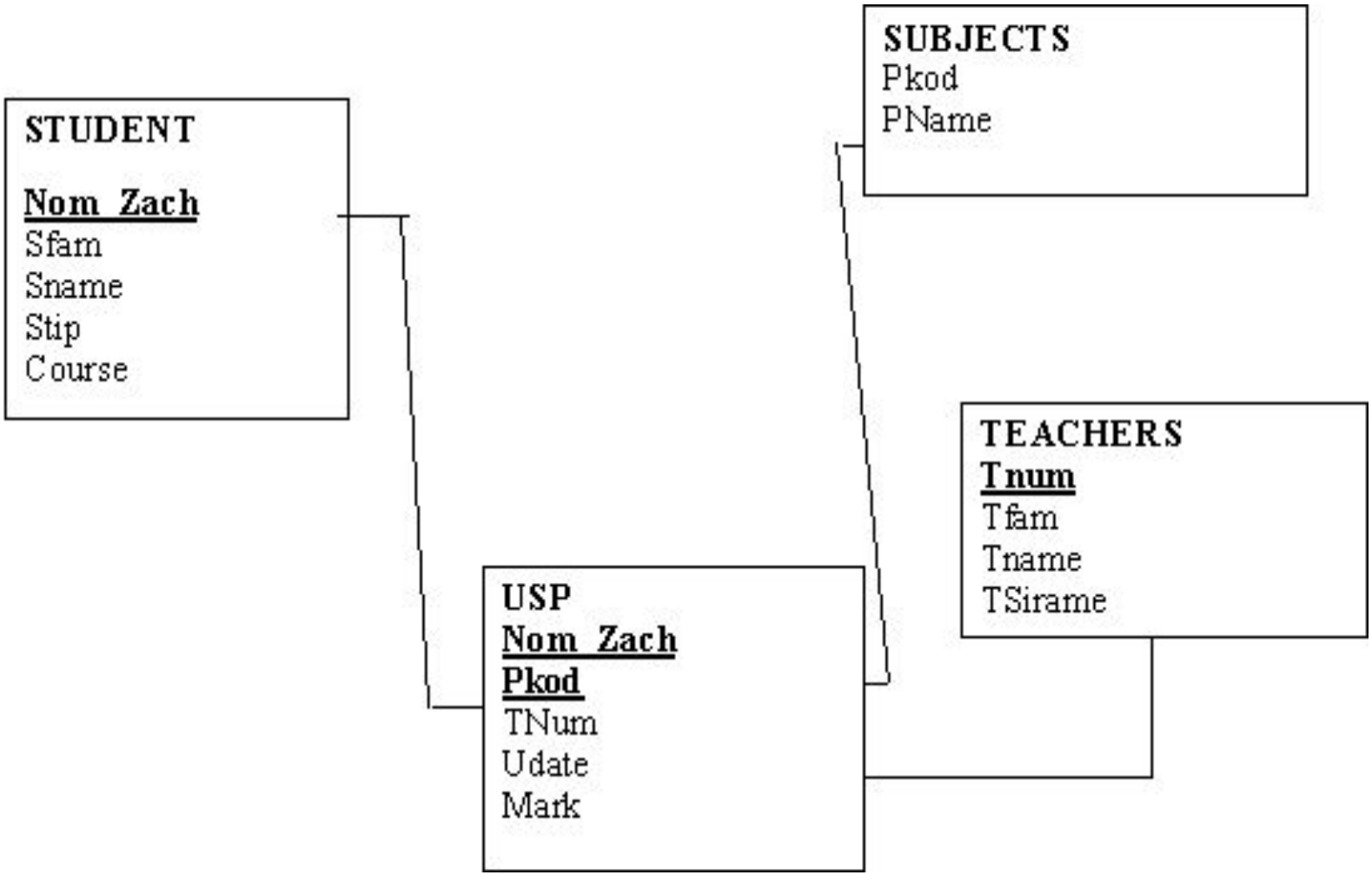
```
CREATE TABLE USP
(NOM_ZACH INTEGER NOT NULL,
PKOD INTEGER,
TNUM INTEGER,
UDATE DATE ,
MARK INTEGER,
PRIMARY KEY(NOM_ZACH, PKOD)
FOREIGN KEY (PKOD) REFERENCES SUBJECTS
FOREIGN KEY (NOM_ZACH) REFERENCES
   STUDENTS  ON UPDATE CASCADE)
```

Если данные о студенте удаляются, удаление их должно быть выполнено сначала в подчинённой (**USP**), а затем в главной таблице (**STUDENTS**).

В этом случае используется ограничение

ON DELETE NO ACTION

После этого при удалении данных о студенте из таблицы **STUDENT** команда не будет выполнена до тех пор, пока не будут удалены его данные из таблицы **USP**.


SUBJECTS
Pkod
PName
STUDENT
Nom_Zach
Sfam
Sname
Stip
Course
TEACHERS
Tnum
Tfam
Tname
TSirame
USP
Nom_Zach
Pkod
TNum
Udate
Mark

Изменение таблиц

При необходимости в уже созданную таблицу можно внести изменения, например, добавить столбец.

ALTER TABLE STUDENTS ADD S_TEL VARCHAR (20) NOT NULL

Новое поле станет последним по порядку в таблице. Допускается добавление сразу нескольких полей. Они должны быть отделены друг от друга запятыми.

Предполагается, что столбец может содержать неопределенные значения, если в объявлении не указано обратное. Если новый столбец не допускает неопределенных значений, необходимо определить для него значение по умолчанию.

Новые столбцы могут представлять собой вычисляемые выражения и объявляться с ограничениями уровня столбцов.

В таблицу могут быть добавлены и новые ограничения с помощью команды

ADD CONSTRAINT <имя ограничения>.

Имя ограничения состоит из краткого названия типа ограничения (например, **PK** для первичного ключа, **ID** для индекса), символа подчёркивания, имени поля или таблицы и порядкового номера ограничения данного типа, если к одному объекту задаётся несколько ограничений одного типа.

Примеры:

1. Для добавления ограничения, задающего значение по умолчанию:

ALTER TABLE USP
ADD CONSTRAINT Def_Mark DEFAULT 7 FOR MARK

2. Для добавления ограничения проверки значения:

ALTER TABLE USP
ADD CONSTRAINT Ch_Mark CHECK MARK IN (3,4,5)

3. Для добавления внешнего ключа (***NOM_ZACH***) в таблицу ***USP*** для связи с таблицей ***STUDENTS***

ALTER TABLE USP

ADD CONSTRAINT FK_USP_STUDENTS FOREIGN KEY (NOM_ZACH)

REFERENCES STUDENTS ON UPDATE CASCADE

Для получения информации об ограничениях используется системная процедура

sp_helpconstraint *имя_таблицы*

или **sp_help** *имя ограничения.*

Удаление столбцов и ограничений

Из созданной таблицы можно удалить столбцы или ограничения. При удалении ограничений следует помнить, что выполнению команды могут помешать некоторые зависимости.

Например, если столбец является первичным ключом, сервер не позволит удалить его до тех пор, пока не будет снято соответствующее ограничение. Если в другой таблице существует ссылка на столбец, сервер также не позволит удалить его до снятия ограничения.

Примеры:

```
ALTER TABLE USP
 DROP CONSTRAINT   Ch_Mark

ALTER TABLE USP
 DROP COLUMN   Udate
```

Разрешение и запрет ограничений

С помощью команды **ALTER TABLE** с предложениями **ENABLE** и **DISABLE** можно разрешать и запрещать действия ограничений, не удаляя их.

ALTER TABLE таблица

ENABLE | DISABLE CONSTRAINT *имя ограничения*

Модификация столбцов

Иногда при создании таблиц делают неверные предположения относительно типа данных, которые собираются хранить в таблице. Неверный выбор приводит к неэффективному хранению данных или же данные могут оказаться слишком большими и не помещаться в столбцах. В таких ситуациях следует изменить исходное определение столбцов командой ***ALTER COLUMN.***

Пример:

```
ALTER TABLE PREP
ALTER COLUMN FIO varchar(50)
```

Удаление таблиц

Удаление таблиц выполняется с помощью команды **DROP TABLE**.

Для того чтобы иметь возможность удалить таблицу, пользователь должен быть ее владельцем.

Кроме того, перед удалением SQL требует очистки таблицы от данных.

Таким образом, таблица с находящимися в ней данными не может быть удалена. Перед удалением следует убедиться, что на таблицу не ссылается никакая другая таблица, и что она не используется в каком-либо представлении.

Синтаксис команды удаления:

DROP TABLE <имя таблицы>

Создание индексов

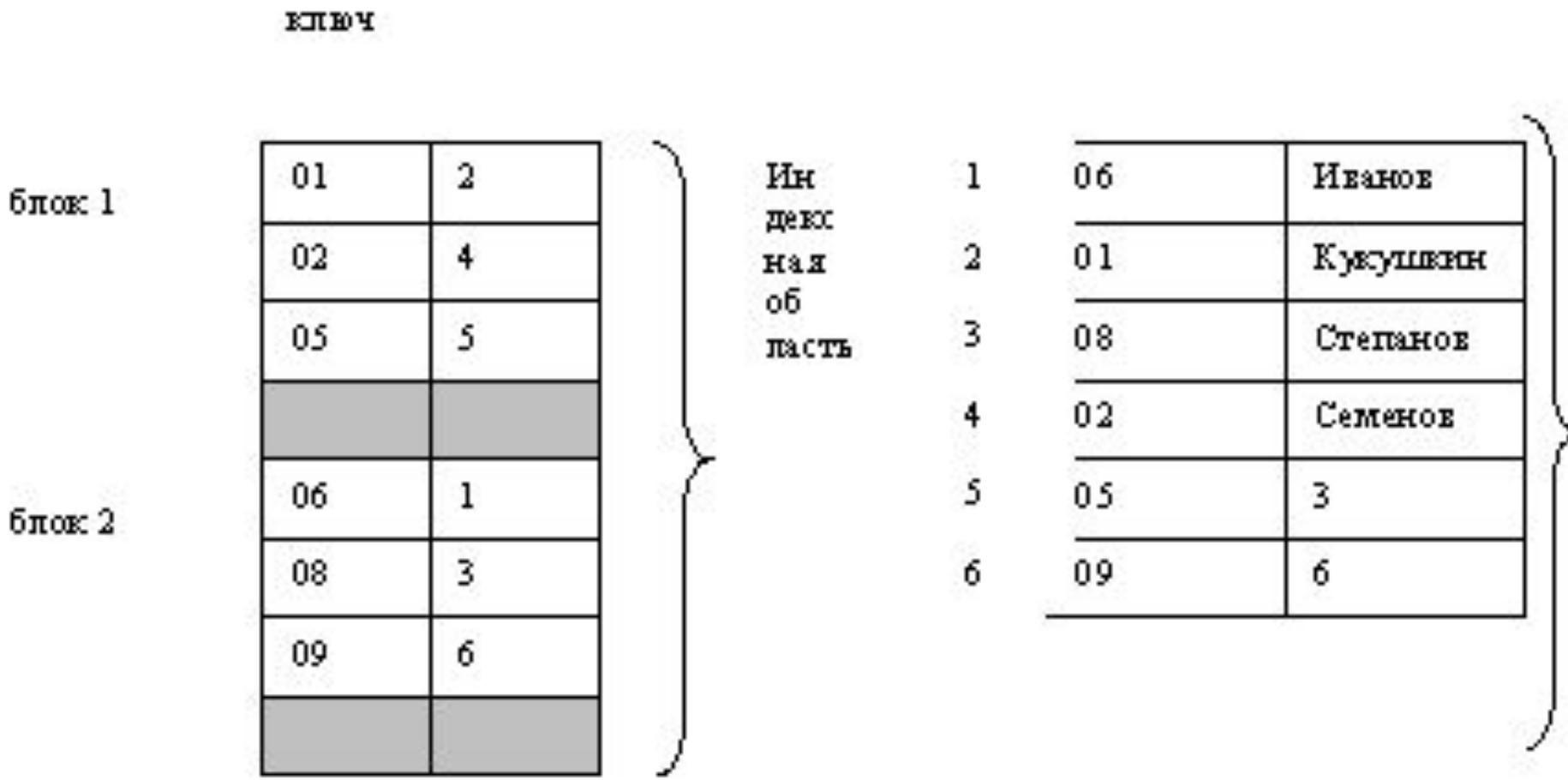
Индекс - упорядоченный список полей или групп полей в таблице.

Таблицы могут иметь огромное количество записей, при этом записи не находятся в каком-либо определённом порядке, поэтому на их поиск по указанному критерию может потребоваться достаточно продолжительное время.

Когда создаётся индекс для поля, база данных запоминает порядок всех значений этого поля в **специальной области памяти**.

При этом **каждое значение поля в индексе содержит ссылку на физическое расположение записи в самой таблице данных.**

Индексы могут состоять из нескольких полей, при этом первое поле является как бы главным, второе упорядочивается внутри первого, третье внутри второго и т.д.


ключ
блок 1
блок 2
01
2
02
4
05
5
06
1
08
3
09
6
Ин
декс
ная
об
ласть
1
2
3
4
5
6
06
Иванов
01
Кукушкин
08
Степанов
02
Семенов
05
3
09
6

С помощью индексов осуществляется доступ к данным наиболее оптимальным способом.

В индексы следует включать поля, к которым часто выполняются запросы при выполнении операций выборки. К ним относятся внешние ключи, а также другие поля, часто используемые для соединения таблиц.

Для создания индекса используется оператор **CREATE INDEX**.

Синтаксис:
CREATE INDEX *имя_индекса* ***ON*** *таблица (поле[, …n])*

Таблица, для которой создаётся индекс, должна уже существовать и содержать имена индексируемых полей. При этом имя индекса не может быть использовано для чего-либо другого в базе данных и SQL сам решает, когда он необходим для работы и использует его автоматически.

Для создания уникальных (не содержащих повторяющихся значений) индексов используется ключевое слово **UNIQUE** в операторе **CREATE *INDEX (CREATE UNIQUE INDEX …).***

Например, создать индекс с именем ***Ind_Tnum*** для поля **TNUM** таблицы **SUBJECT:**

CREATE UNIQUE INDEX Ind_Tnum ON SUBJECT(TNUM)

Для удаления индекса используется команда

DROP INDEX имя индекса

Чтобы изменить индекс таблицы, необходимо удалить его и затем создать заново в соответствии с новым определением.

Для получения информации о текущих индексах таблицы используется процедура ***sp_helpindex.***

Кластеризованный индекс

Использование опции **Clustered index** позволяет произвести так называемое кластерное индексирование, в результате чего будут отсортированы данные в самой таблице согласно порядку этого индекса, и вся добавляемая информация будет приводить к изменению физического порядка данных. При этом нужно учитывать, что в таблице может быть определён только один кластерный индекс.

Синтаксис:
CREATE CLUSTERED INDEX

Для повышения быстродействия кластерный индекс следует создавать раньше некластерных индексов.

По умолчанию создается некластерный индекс.