

# Реляционная модель

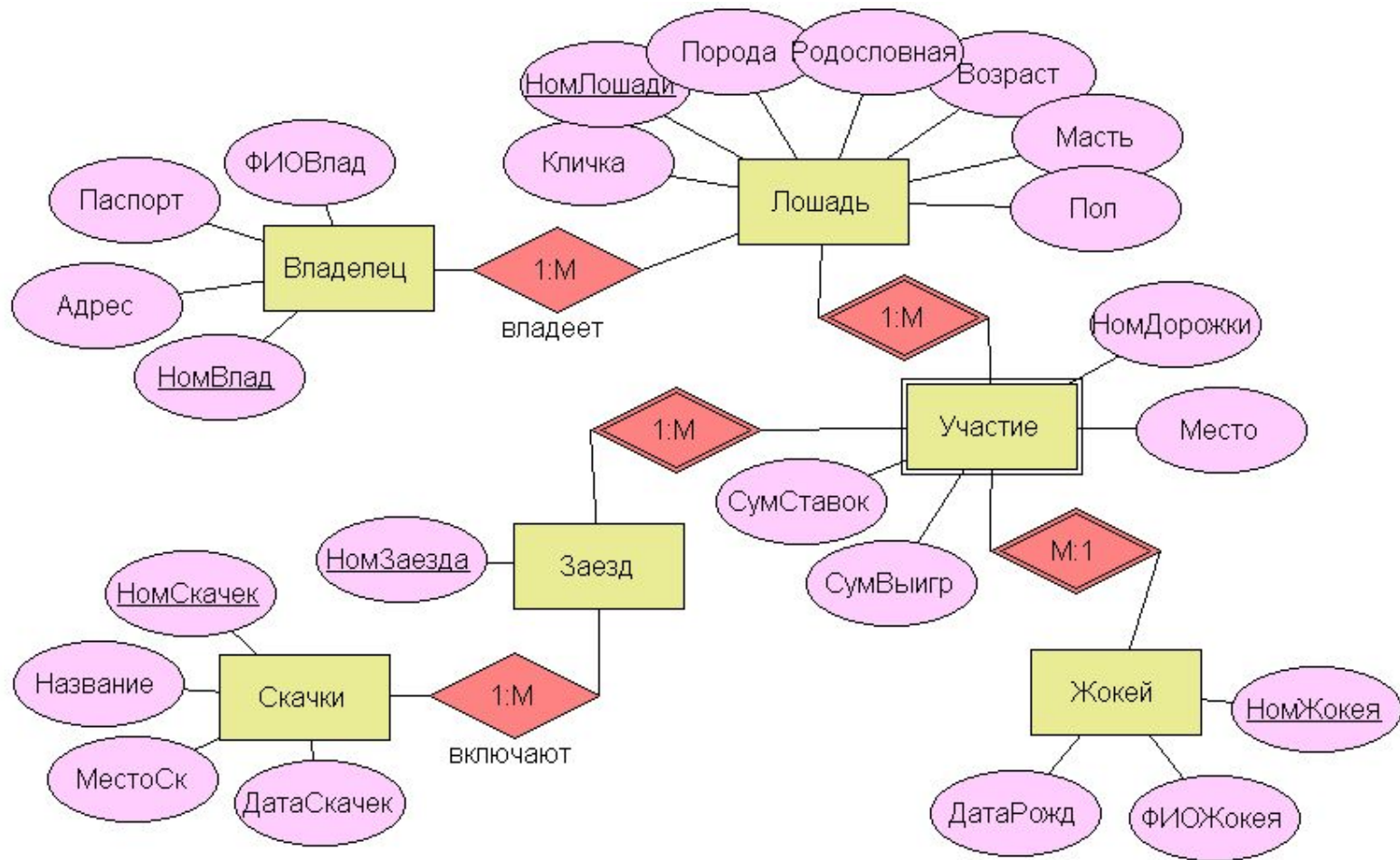
# Модель данных

- Определяет абстракцию данных для приложений
- Включает:
  - Структуры данных
  - Операции
  - Зависимости
  - Ограничения

# Модель данных «Сущность-связь»

- Впервые описана в 1975 г. как средство высокоуровневого проектирования БД
- Инструмент концептуального уровня
- Не содержит операций и поэтому не может использоваться непосредственно
- Развитая система зависимостей и ограничений целостности

# Пример ER-диаграммы



# Концептуальные модели данных

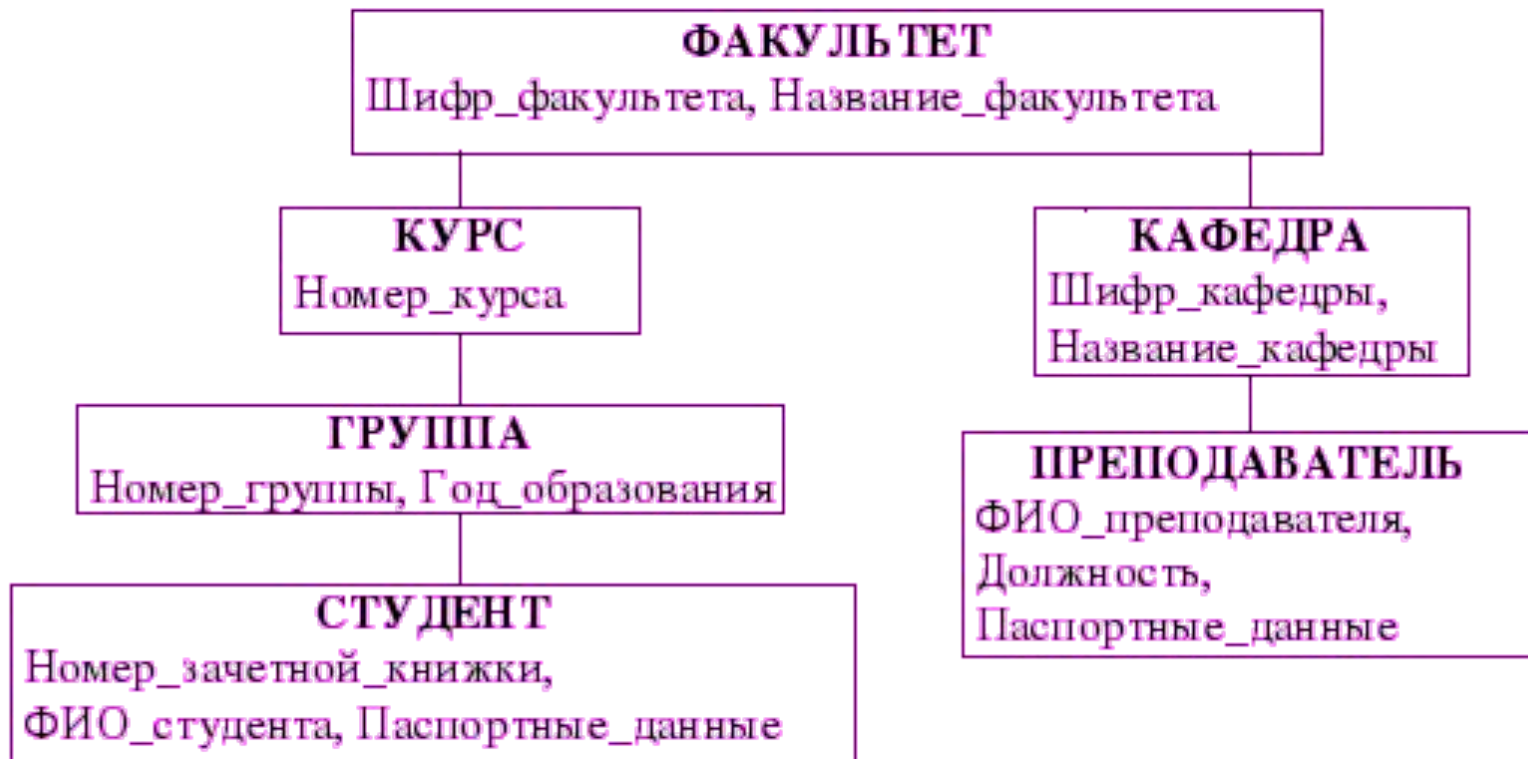
- Иерархическая
- Сетевая
- Реляционная
- Объектно-реляционная

В 70-х – 80-х годах, когда компьютерные ученые все еще носили коричневые смокинги и очки с большими, квадратными оправками, данные хранились бесструктурно в файлах, которые представляли собой текстовый документ с данными, разделенными (обычно) запятыми или табуляциями.

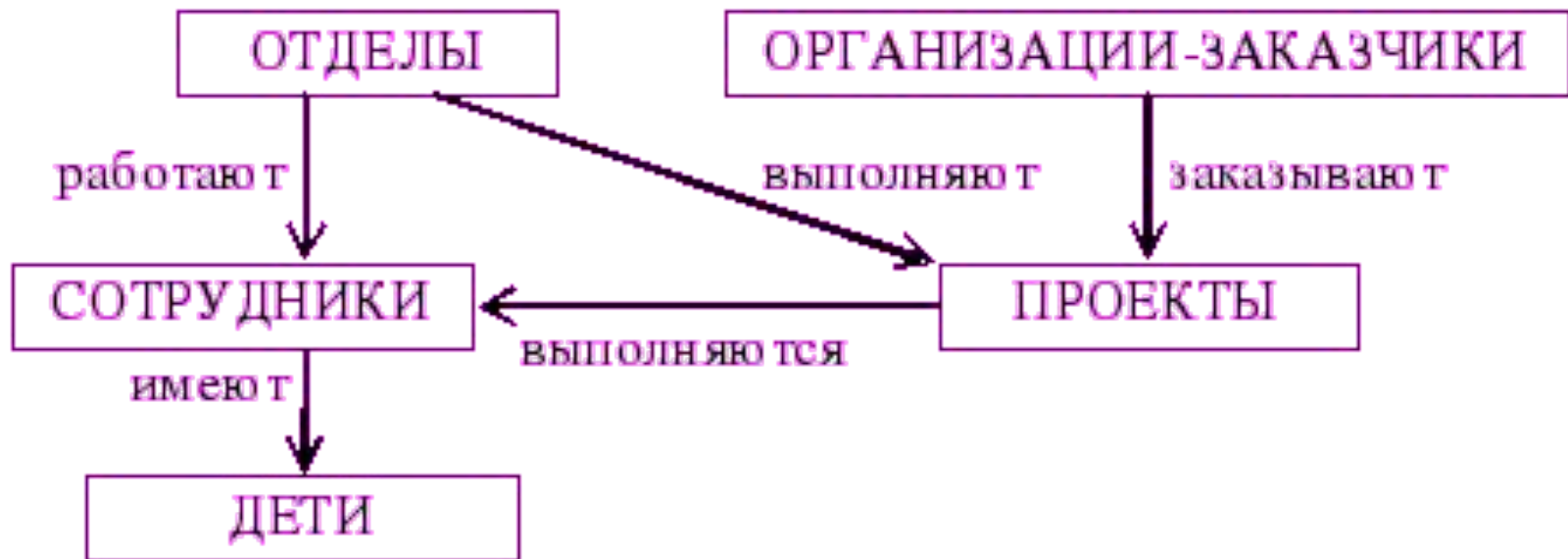


```
'tutorial_id', 'title', 'category'  
'1', 'Access Tutorial', 'Software'  
'2', 'Excel Tutorial', 'Software'  
'3', Database design tutorial, 'Software'  
'4', 'Oracle DBA Course ', 'Software'  
'5', 'Raid Storage Tutorial ', 'Hardware'  
'6', 'Network Security Tutorial', 'Networks'
```

# Иерархическая модель данных



# Сетевая модель данных



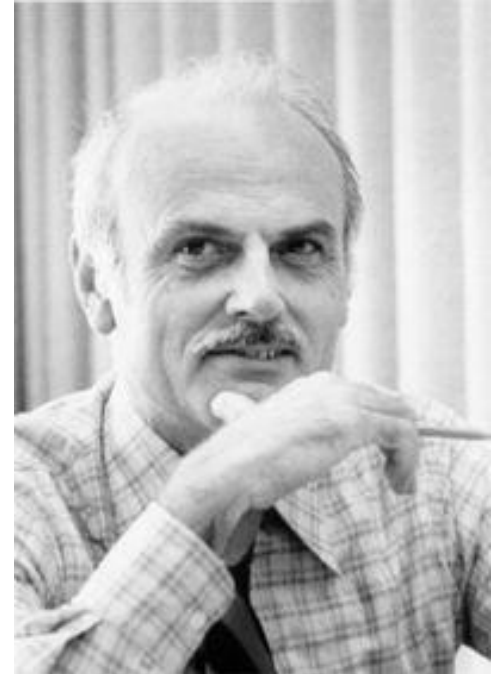


# Реляционная модель

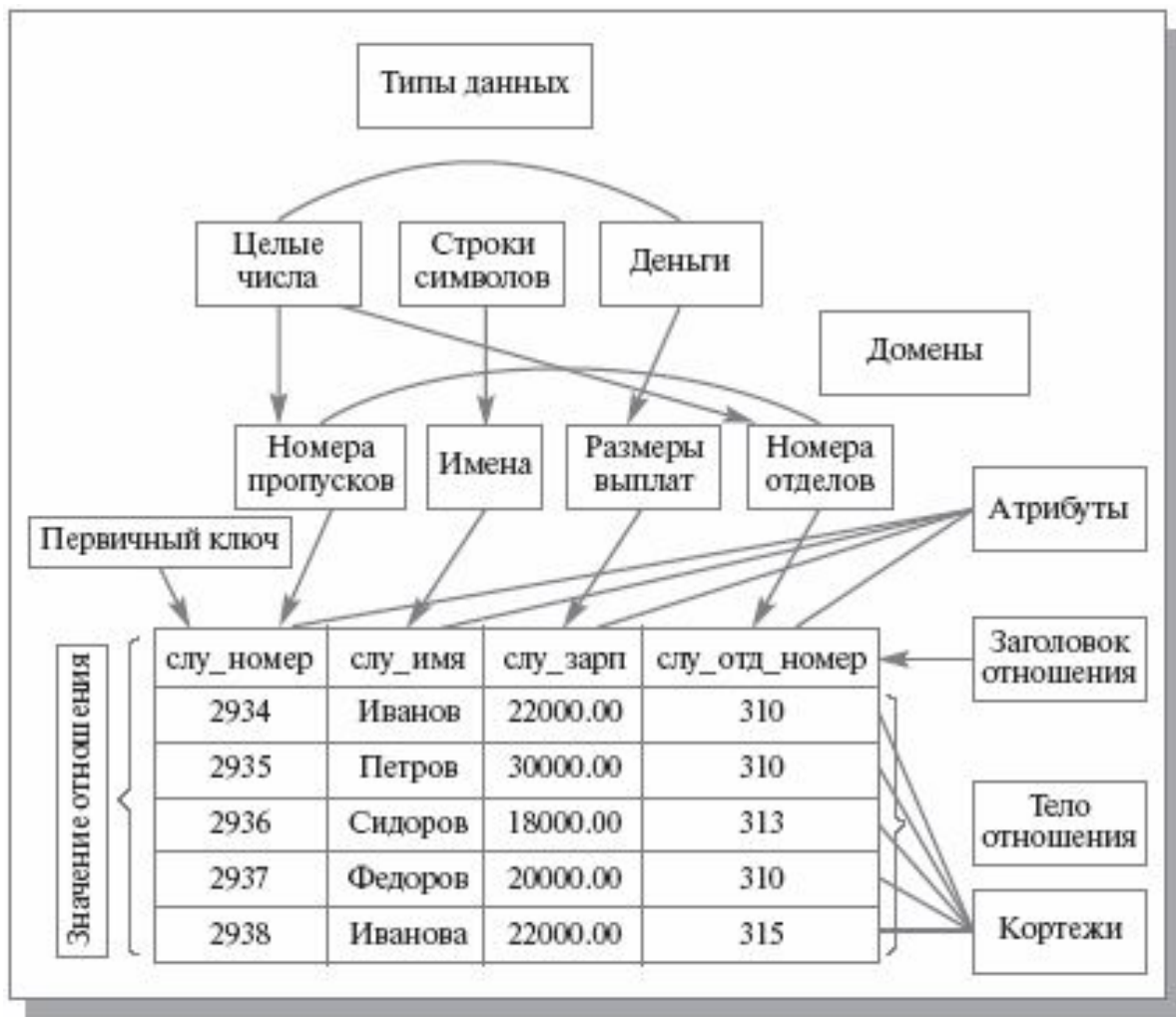
- Первые публикации (1969-1971)
- Интенсивное развитие теории в 70-х гг.
- Ранние попытки реализации: 1978
- Стандарт SQL –1986
- Эффективные реализации SQL: 1990

# Эдгар Франк Кодд

- Codd, E.F. (1970).  
«A Relational Model of Data  
for Large Shared Data Banks».  
Communications of the ACM



# Пример



# Реляционная модель – структуры данных

- Домены: множества, элементы которых рассматриваются как скалярные значения
- Отношения: предикаты, заданные на прямом произведении (не обязательно разных) доменов
- Атрибуты: аргументы отношений
- Позиционные или именованные атрибуты?

# Реляционная модель

- Домены  $D_1, D_2, \dots, D_n$
- Атрибуты  $A_1, A_2, \dots, A_n$
- Кортежи  $t = \langle a_1, a_2, \dots, a_n \rangle$ ,  $a_i \in D_i$
- Формально  $R : D_1 \times D_2 \times \dots \times D_n \rightarrow \{0, 1\}$   
или  $R \subset D_1 \times D_2 \times \dots \times D_n$
- $D_1 = \{1; 2; 3\}; D_2 = \{a, b\}$
- $R \subset D_1 \times D_2 =$   
 $\{\langle 1; a \rangle; \langle 1; b \rangle; \langle 2; a \rangle; \langle 2; b \rangle; \langle 3; a \rangle; \langle 3; c \rangle\}$

# Реляционная модель

- Похожа на таблицы
- Столбцы – атрибуты
- Строки – данные
- Шапка таблицы – имена атрибутов

# Еще раз терминология

- Домен – множество возможных значений какого-либо атрибута
- Таблица - отношение
- Экземпляр– конкретное наполнение базы данных
- Конкретное наполнение таблицы – тело отношения
- Совокупность атрибутов отношения – Заголовок отношения (схема)
- Заголовок + тело = значение отношения
- Строка таблицы называется кортежем

# Заголовок (схема) отношения

- Схема отношения - конечное множество упорядоченных пар вида  $\langle A, T \rangle$ , где  $A$  называется именем атрибута, а  $T$  обозначает имя некоторого базового типа или ранее определенного домена.
- По определению требуется, чтобы все имена атрибутов в заголовке отношения были различны.
- Количество атрибутов называется арностью (размерностью) отношения.



# Типы данных

- Символьный
- Битовый
- Точные числа
- Округленные числа
- Денежные
- Дата/время
- Интервал

# Отношение – это множество

- Кортежи отличаются друг от друга значением своих атрибутов, а не порядковым номером, временем и пр.
- В реляционной БД не может быть двух одинаковых кортежей в одной таблице

# Сущности

- Каждая сущность превращается в таблицу. Имя сущности становится именем таблицы.
- Каждый атрибут становится столбцом. Столбцы для необязательных атрибутов могут содержать неопределенные значения; столбцы для обязательных - не могут.
- Компоненты уникального идентификатора сущности превращаются в ключ таблицы.

# СУЩНОСТИ



Сущность

| Ключ | Атрибут 1 | Атрибут 2 |
|------|-----------|-----------|
|      |           |           |

# Ключ

- Возможный ключ: минимальный набор атрибутов, от которого функционально зависят все остальные (по которому можно определить все остальные)
- Первичный ключ: один из возможных ключей
- Ключи естественные и суррогатные

# СВЯЗИ

- Также хранятся в отношении
- Схема данного отношения составляется из ключевых атрибутов объектов, участвующих связи

# Связи 1:1



Отдел

| <u>Табельный номер</u> | ФИО | <u>Номер отдела</u> | Название_отдела |
|------------------------|-----|---------------------|-----------------|
|                        |     |                     |                 |

# СВЯЗИ 1:N



Отдел

| <u>Номер_отдела</u> | Название_отдела |
|---------------------|-----------------|
|                     |                 |

Сотрудник

| <u>Табельный_номер</u> | ФИО | Номер_отдела |
|------------------------|-----|--------------|
|                        |     |              |



# Слабые сущности



Сотрудник

|                        |     |
|------------------------|-----|
| <u>Табельный номер</u> | ФИО |
|                        |     |

Адрес

|                        |              |            |
|------------------------|--------------|------------|
| <u>Табельный номер</u> | <u>Улица</u> | <u>Дом</u> |
|                        |              |            |

# Связи М:N



Проект

| <u>Номер_проекта</u> | Название_проекта |
|----------------------|------------------|
|                      |                  |

Сотрудник

| <u>Табельный_номер</u> | ФИО |
|------------------------|-----|
|                        |     |

Участие в  
проекте

| <u>Табельный_номер</u> | <u>Номер_проекта</u> |
|------------------------|----------------------|
|                        |                      |

# Реляционная алгебра – механизм манипулирования реляционными данными

Все операции производятся над отношениями, и результатом операции является отношение.

$$R=f(R_1, R_2, \dots, R_n)$$

# Реляционная модель: базовые операции

- Ограничение (селекция, фильтрация)
- Проекция
- Прямое (декартово) произведение
- Соединение: произведение с последующей фильтрацией
- Естественное соединение: соединение по равенству атрибутов

# SQL

## Structured Query Language

Structured English Query Language (1983)

Стандарты: 86, 89, 92, 1999

- DDL

create table, create view, . . . , alter table,  
alter view, . . . , drop table, drop view, . . .

- DML

select, insert, delete, update, commit, rollback

# Создание базы данных

- `CREATE DATABASE db_name;`
- `DROP DATABASE db_name;`
- `USE db_name;`

# Создание БД

- `CREATE DATABASE database_name [options]`

# Идентификаторы

- идентификатор может иметь длину до 128 символов;
- идентификатор должен начинаться с буквы;
- идентификатор не может содержать пробелы.
  - номер int,
  - [Фамилия и Имя] nvarchar(100),
  - [!@#\$%^&\*()\_-=+ ~`\. ,;:'"[]<>] int);



```
CREATE DATABASE Sales
ON ( NAME = Sales_dat, FILENAME =
    'C:\tmp\saledat.mdf',
    SIZE = 10,
    MAXSIZE = 50,
    FILEGROWTH = 5 )
LOG ON ( NAME = Sales_log, FILENAME =
    'C:\tmp\salelog.ldf',
    SIZE = 5MB,
    MAXSIZE = 25MB,
    FILEGROWTH = 5MB );
```

# О синтаксисе

- Команды разделяются знаком «;»
- Пакеты команды разделяются словом «go»
- Комментарии:
  - В одной строке «--»
  - Многострочные «/\* ... \*/»

# Объекты базы данных

- Таблицы
- Связи между таблицами
- Представления
- Индексы
- Хранимые процедуры
- Функции
- Триггеры

# Как создать таблицу

```
CREATE TABLE tb_name (  
    ( column1 <datatype>,  
      column2 <datatype>, ... )  
);
```

# Типы данных

- Точные числа
- Округленные числа
- Символьный
- Символьный Юникода
- Битовый
- Дата/время

# Точные целые

- `bigint` 8 байт
- `int` 4 байта
- `smallint` 2 байта
- `tinyint` 1 байт

# Точные десятичные

- `decimal[ (p[ ,s] ) ]` и `numeric[ (p[ ,s] ) ]`  
Числа с фиксированной точностью и масштабом.
- `p` (точность)  
Максимальное количество десятичных разрядов числа (как слева, так и справа от десятичной запятой). Точность должна принимать значение от 1 до 38. По умолчанию для точности принимается значение 18.
- `s` (масштаб)  
Максимальное количество десятичных разрядов числа справа от десятичной запятой. Масштаб может принимать значение от 0 до `p`. Масштаб может быть указан только совместно с точностью. По умолчанию масштаб принимает значение 0; поэтому  $0 \leq s \leq p$ . Максимальный размер хранилища зависит от точности.

# Decimal/Numeric

- «Неупакованное» число с плавающей точкой. Ведет себя подобно строковому столбцу, содержащему цифровое значение.
- Термин «неупакованное» означает, что число хранится в виде строки и при этом для каждого десятичного знака используется один символ.
- Разделительный знак десятичных разрядов, а также знак '-' для отрицательных чисел не учитываются в `p` (но место для них зарезервировано). Если атрибут `s` равен 0, величины будут представлены без десятичного знака, т.е. без дробной части.



# Decimal/Numeric

- Для хранения чисел, строго меньших 1000 с двумя знаками после запятой используем `decimal(5,2)`

# Точные денежные

- Типы данных, представляющие денежные (валютные) значения.

| Тип данных              | Диапазон   | Хранилище |
|-------------------------|--|-----------|
| <code>money</code>      | От -922 337 203 685 477,5808 до 922 337 203 685 477,5807 | 8 байт    |
| <code>smallmoney</code> | От -214 748,3648 до 214 748,3647                         | 4 байта   |

- Типы данных `money` и `smallmoney` имеют точность до одной десятитысячной денежной единицы, которую они представляют.
- Перед числовым значением может указываться символ валюты, но храниться будет только число.

# Bit - БИТОВЫЙ

- bit

Целочисленный тип данных, который может принимать значения 1 или 0.

# Округленные числа

- Float и real
- Float – точность до 15 знаков
  - - 1,79E+308 — -2,23E-308, 0 и 2,23E-308 — 1,79E+308
  - 8 байтов
- real - точность до 7 знаков
  - - 3,40E + 38 — -1,18E - 38, 0 и 1,18E - 38 — 3,40E + 38
  - 4 байта

# Float/real

- Избегайте использования столбцов типов **float** и **real** для сравнения с операторами = и <>.
- Рекомендуется ограничить использование столбцов **float** и **real** операциями сравнения > или <.
- Microsoft SQL Server использует округление вверх.

# СИМВОЛЬНЫЕ

- `char [ ( n ) ]`

Символьные данные фиксированной длины, не в Юникоде, с длиной  $n$  байт. Значение  $n$  должно находиться в интервале от 1 до 8000. Размер хранения данных этого типа равен  $n$  байт.

- `varchar [ ( n | max ) ]`      `text` ↔ `varchar(max)`

Символьные данные переменной длины, не в Юникоде.  $n$  может иметь значение от 1 до 8 000. `max` означает, что максимальный размер хранения равен  $2^{31}-1$  байт (2 ГБ). Размер хранения равен фактической длине данных плюс два байта.

Не чувствительны к регистру!

# Символьные Unicode

- `nchar [ ( n ) ]`

Символьные данные фиксированной длины в Юникоде, с длиной  $n$  символов. Значение  $n$  должно находиться в интервале от 1 до 4000. Размер хранения данных этого типа равен  $2n$  байт.

- `nvarchar [ ( n | max ) ]`

Символьные данные переменной длины в Юникоде.  $n$  может иметь значение от 1 до 4 000.

`max` означает, что максимальный размер хранения равен  $2^{31}-1$  байт (2 ГБ). Размер хранения равен фактической длине данных плюс два байта.

Не чувствительны к регистру!

# Ввод русских букв в таблицу

- Тип поля на `nvarchar(N)` или `nchar(N)`
- Перед тестом поставить букву N  
(N'русский текст')



# БИТОВЫЕ

- `binary [ ( n ) ]`

Битовые данные фиксированной длины, с длиной  $n$  байт. Значение  $n$  должно находиться в интервале от 1 до 8000. Размер хранения данных этого типа равен  $n$  байт.

- `varbinary [ ( n | max ) ]`

Битовые данные переменной длины.  $n$  может иметь значение от 1 до 8 000.

$max$  означает, что максимальный размер хранения равен  $2^{31}-1$  байт (2ГБ). Размер хранения равен фактической длине данных плюс два байта.

Чувствительны к регистру!

# Очень большие

- `ntext`, `text`, и `image` данные будут удалены в будущей версии Сервера MicrosoftSQL.
- Избегайте использования ЭТИХ типов данных
- Используйте `nvarchar (max)`, `varchar (max)`, и `varbinary (max)` вместо этого.

# DATE/TIME

- Date - 3 байта  
'YYYY-MM-DD'
- Time - 5 байт  
Время. Интервал от '-838:59:59' до  
'838:59:59' 'HH:MM:SS'
- Datetime - 8 байт  
'YYYY-MM-DD HH:MM:SS'
- Datetimeoffset - 10 байт (+ часовой пояс)

# DATE/TIME

- Date and time(8 байт) - Дата и время с точностью в 3.33 миллисекунды.
- Smalldatetime(4 байта) - Дата и время с точностью в 1 секунду.

# DATE/TIME

- От 0001-01-01 до 9999-12-31
- От 1 января 1 года нашей эры до 31 декабря 9999 года нашей эры.

# Специальные типы данных

- **hierarchyid** - позиция в древовидной иерархии
- **geometry** представляет данные в Евклидовой (плоской) системе координат.
- **geography** представляет данные в системе координат круглой земли (координаты широты и долготы в системе GPS).
- **xml** - хранение XML-данных

# Константы

- NULL
- 'any string'
- N'текст в Юникоде'
- Числа без кавычек
- '1912-10-25 12:24:32 +10:0'

# Определение типов данных атрибутов - строки

- Для коротких символьных значений и строк фиксированной длины следует выбирать тип CHAR. Например, для поля "единица измерения" со значениями 'кг', 'шт.', 'уп.' (char(3)), для поля "пол" (char(1)) и т.п.
- Для символьных строк переменной длины нужно выбирать тип VARCHAR с указанием максимально возможной длины хранимого значения.
- Для символов на 3 и более языках nchar или nvarchar



# Определение типов данных атрибутов - числа

- Для числовых атрибутов без сложных расчетов – тип `numeric`, указывая реально необходимое количество разрядов.  
Номер сотрудника `numeric(4)` (до 10000 человек)  
Зарплата – `numeric(8, 2)` (до 999999.99 рублей).
- Для числовых атрибутов с расчетами числовые типы, которые хранят данные в машинном (двоичном) представлении (`int`, `bigint`...).  
Это ускорит выполнение расчётов.
- Для числовых атрибутов, имеющих ведущие нули, тип `CHAR`, а не числовой тип, иначе ведущие нули будут потеряны.  
Серия и номер паспорта (`char(10)`).

# Определение типов данных атрибутов - прочее

- Для хранения дат нужно выбирать тип DATE или его варианты (DATETIME, например). Это позволит использовать арифметику дат и не заботиться о правильности вводимых данных: СУБД сама проверит допустимость даты.
- Для семантически одинаковых полей разных таблиц нужно выбирать одинаковые типы данных. Например, ФИО сотрудника и ФИО клиента.
- Для упрощения типизации данных можно создать специальные типы данных (create type) и использовать их в качестве типов полей таблиц.

# Типы данных, определяемые пользователем

```
CREATE TYPE mytype  
FROM varchar(11) NOT NULL ;
```

# Табличный тип

```
CREATE TYPE Location AS TABLE  
  ( LocationName VARCHAR(50)  
    , CostRate INT );
```

# Преобразование типов данных

- `SELECT '2'+4`
- Функции `CAST` и `CONVERT`
- `CAST ( expression AS data_type )`
- `CONVERT ( data_type, expression [ , style ] )`

# Примеры CAST

- CAST (expression AS data\_type)
- SELECT CAST('12' as INT)+45  
57
- SELECT CAST(27 as CHAR(2))+ 'R'  
27R
- SELECT CAST ( \$10.50 as VARCHAR(10) )  
10.50

# Примеры CONVERT

- `CONVERT ( data_type, expression [ , style ] )`
- `CONVERT(VARCHAR(12), GETDATE(), 1)`  
`02/21/15`
- `SELECT CONVERT (bit, 'true')`  
`1`
- `SELECT CONVERT(bit, 'false')`  
`0`

# Временные таблицы локальные

```
CREATE TABLE #TestTable  
(  
  id INT PRIMARY KEY  
)
```

Таблица будет существовать только во время выполнения одной сессии, и работать с ней сможете только вы.

БД tempdb



# Временные таблицы глобальные

```
CREATE TABLE ##TestTable  
(  
  id INT PRIMARY KEY  
)
```

Таблица будет видна всем. Уничтожается после закрытия создавшей ее сессии /окончания работы с ней другими пользователями

# Табличные переменные

- `DECLARE @table_var table(id int);`
- Автоматически очищаются в конце функции, хранимой процедуры или пакета, где они были определены
- Табличная переменная не участвует в транзакции.
- Не подходят для хранения значительных объёмов данных (>100 строк).

# Передача имени таблицы

```
DECLARE @SQL varchar(8000),  
@table_name varchar(20)='dbo.Employees'  
  
SET @SQL = 'SELECT * FROM ' + @table_name  
exec(@SQL)
```

# Типы данных, определяемые пользователем

user defined type

```
CREATE TYPE mytype  
FROM varchar(11) NOT NULL ;
```

# Табличный тип

- В БД объявляется user defined type в виде таблицы с нужным типом данных. Его можно передавать в хранимую процедуру как параметр. В хранимой процедуре этот тип будет виден как t-sql таблица, к которой можно делать запросы.
- `CREATE TYPE newlist AS TABLE(id int NULL)`

# Создание таблицы

- CREATE TABLE table\_name  
(  
atr1, [atr2,...]  
)

# Определение атрибутов таблицы

- { имя столбца }
- { тип данных }
- [ значение по умолчанию ]
- [ список ограничений ]

# Создание таблицы

- CREATE TABLE people (  
id INT  
, name CHAR(20)  
, address VARCHAR(35)  
, birthday DATE)



# Задание ограничений

- NOT NULL
- | [ PRIMARY KEY | UNIQUE ]
- | references
- | CHECK ( условие )

# Ненулевые значения

- CREATE TABLE people (  
id INT  
, name CHAR(20) NOT NULL  
, address VARCHAR(35)  
, birthday DATE)

# Значение по умолчанию

- CREATE TABLE people (  
id INT  
, name CHAR(20)  
, address VARCHAR(35) DEFAULT 'Без регистрации'  
, birthday DATE)

# Уникальность поля

- CREATE TABLE people (  
id INT UNIQUE NULL  
, name CHAR(20)  
, address VARCHAR(35)  
, birthday DATE)
- Несколько полей (или их комбинаций) могут быть уникальны
- Уникальное поле может быть NULL

# Первичный ключ

- CREATE TABLE people (  
id INT PRIMARY KEY  
, name CHAR(20) NOT NULL  
, address VARCHAR(35)  
, birthday DATE);
- PRIMARY KEY – всегда NOT NULL
- PRIMARY KEY – только один в таблице

# Первичный ключ – несколько атрибутов

- CREATE TABLE people (  
id INT UNIQUE  
, name CHAR(20)  
, address VARCHAR(35)  
, birthday DATE  
, PRIMARY KEY (name, birthday))

# Уникальность – несколько атрибутов

- CREATE TABLE people (  
id INT PRIMARY KEY  
, name CHAR(20)  
, address VARCHAR(35) UNIQUE  
, birthday DATE  
, UNIQUE (name, birthday)),

# Вычисляемый столбец

- CREATE TABLE t1 (  
a int,  
b int,  
c AS a/b [ PERSISTED [ NOT NULL ] ] );



# Вычисляемый столбец

- Если не указано иное, вычисляемые столбцы являются виртуальными столбцами, которые физически не хранятся.
- Их значения вычисляются заново каждый раз при обращении к ним запроса.
- Для физического хранения вычисляемых столбцов в таблицах используется ключевое слово PERSISTED - можно создать на вычисляемом столбце индекс

# Вычисляемый столбец

- Если формула связывает два выражения различных типов данных, то по правилам приоритета типов данных определяется, какой тип данных имеет меньший приоритет и будет преобразован в тип данных с большим приоритетом.
- Если неявное преобразование не поддерживается, возвращается ошибка «Error validating the formula for column column\_name.».
- Используйте функцию CAST или CONVERT, чтобы устранить конфликт типа данных.
- $6/4=1$  !!!!!

# Вычисляемые столбцы при создании таблицы

```
CREATE TABLE t2 (  
  a int  
  , b int  
  , c int  
  , x float  
  , y AS CASE x  
            WHEN 0 THEN a  
            WHEN 1 THEN b  
            ELSE c  
            END  
);
```

# Вычисляемые столбцы при создании таблицы

```
CREATE TABLE people (  
    id INT PRIMARY KEY  
    , birth DATE  
    , salary INT  
    , age as DATEDIFF(YEAR, GETDATE(),birth)  
    , nalog as CAST(salary as real)*13/100  
)
```

# Атрибут IDENTITY

- IDENTITY [ (seed ,increment) ]
  - Seed - номер первой строки, вставляемой в таблицу
  - Increment - «Шаг» после последнего добавленного
- Надо определять либо и начальный номер, и шаг, или ничего. По умолчанию (1,1).
- Сброс - DBCC CHECKIDENT ('имя\_таблицы', RESEED, новое\_стартовое\_значение)

# Атрибут IDENTITY

- CREATE TABLE animals  
(id INT NOT NULL IDENTITY(10,2)  
, name CHAR(30) NOT NULL  
, PRIMARY KEY (id));
- INSERT INTO animals (name) VALUES  
("dog"),("cat"),("penguin"),("lax"),("whale");

# Проверка ограничений

- CREATE TABLE people (  
id int PRIMARY KEY  
, name CHAR(20)  
, address VARCHAR(35)  
, birthday DATE  
, beg\_date DATE  
, gender CHAR  
, CHECK (gender IN ('F', 'M'))  
)
- CHECK с использованием функций:  
(стаж(birthday, beg\_date) < возраст(birthday))

# Проверка ограничений

- CREATE TABLE people (  
id int PRIMARY KEY  
, name CHAR(20)  
, address VARCHAR(35)  
, birthday DATE  
, gender CHAR  
, CONSTRAINT chk\_Person CHECK (gender IN  
(‘F’, ‘M’))  
)



# Проверка ограничений шаблоны LIKE

- CREATE TABLE people (  
id int PRIMARY KEY,  
name CHAR(20) CHECK (name LIKE '[A-Z]%),  
zip CHAR(6) CHECK (zip LIKE '\_[1-35][0-9]'),  
address VARCHAR(35),  
birthday DATE,  
gender CHAR  
)

# LIKE

| Символ-шаблон     | Описание   | Пример        |
|-------------------|--|---------------|
| %                 | Любая строка длиной от нуля и более символов.  | '%компьютер%' |
| _ (подчеркивание) | Любой одиночный символ.  | '_етров'      |
| [ ]               | Любой одиночный символ, содержащийся в диапазоне ([a-f]) или наборе ([abcdef]).      | '[Л-С]омов'   |
| [^]               | Любой одиночный символ, не содержащийся в диапазоне ([^a-f]) или наборе ([^abcdef]). | 'ив[^a]%'     |

# Внешний ключ

- CREATE TABLE people (  
id INT PRIMARY KEY  
, name CHAR(20) NOT NULL  
, address VARCHAR(35)  
, birthday DATE);
- У каждого может быть несколько телефонов

# Внешний ключ

- CREATE TABLE phone (  
id\_p int  
, type CHAR CHECK (type IN ('H', 'M', 'W'))  
, phone\_code char(3) DEFAULT '812'  
, phone\_number numeric(7)  
)

# Внешний ключ

- CREATE TABLE phone (  
id\_p int REFERENCES people (id)  
, type CHAR CHECK (type IN ('H', 'M', 'W'))  
, phone\_code char(3) DEFAULT '812'  
, phone\_number char(7)  
)

# Внешний ключ

- CREATE TABLE phone (  
id\_p int,  
type CHAR CHECK (type IN ('H', 'M', 'W')),  
phone\_code char(3) DEFAULT '812',  
phone\_number char(7),  
FOREIGN KEY (id\_p) REFERENCES people (id))

# Требования внешнего ключа

- Поле, на который ссылается внешний ключ, должно быть PRIMARY KEY или UNIQUE
- Оба поля должны быть строго одного типа!
- Называться поля могут по-разному

# Внешний ключ

FOREIGN KEY (index\_col\_name, ...)

REFERENCES table\_name (index\_col\_name, ...)

[ON DELETE

{NO ACTION | SET NULL | CASCADE | SET DEFAULT}]

[ON UPDATE

{NO ACTION | SET NULL | CASCADE | SET DEFAULT}]



# Ссылочная целостность – стратегии:

- NO ACTION - отмена операции
- CASCADE - каскадная модификация  
UPDATE(1) ⇒ UPDATE(2)  
DELETE(1) ⇒ DELETE(2)
- SET NULL - отложенная проверка ограничений – замена несуществующего значения на NULL
- SET DEFAULT замена несуществующего значения на значение по умолчанию

# SET NULL | CASCADE

- CREATE TABLE phone {  
id\_p int REFERENCES people (id)  
ON DELETE CASCADE  
ON UPDATE CASCADE,  
...}

# Внешний ключ – нереализуемые связи

- Нельзя реализовать при помощи внешнего ключа связи с кардинальностью 1:n или n:m, обязательные в обе стороны.
- Например, связь заказы–строки заказов: заказ не может быть пустым, и заказанный товар должен входить в определённый заказ.
- Эта проблема обычно решается так: связь делается необязательной со стороны первичного ключа, а внешний ключ остаётся обязательным. При этом в приложении необходимо предусмотреть правило обработки пустых заказов (например, их удаление).

# Описание ограничений целостности

- Если какое-либо ограничение целостности может быть включено в структуру БД, то его надо реализовать именно так.
- Необходимо обратить особое внимание на поля таблиц, для которых домен определён как список возможных значений. Это ограничение целостности можно реализовать в виде: CHECK(<поле> IN (<список значений>)).
- 'незаконченное высшее',  
'незаконч. высшее',  
'н. высш.' и т.д..

# Описание ограничений целостности

- Но такой подход имеет следующий недостаток: добавление нового значения в список потребует изменения схемы отношения.
- Можно поступить по-другому: вынести этот список значений в отдельное отношение. Например, список типов образования (начальное, неполное среднее, среднее, средне-специальное, незаконченное высшее, высшее) для таблицы СОТРУДНИКИ.
- Таблица ТИПЫ ОБРАЗОВАНИЯ будет состоять из одного поля Название типа, определённого как первичный ключ. Тогда поле Образование таблицы СОТРУДНИКИ станет внешним ключом.

# Внешний ключ – нереализуемые связи

- К необычным конструкциям данных можно отнести так называемые взаимоисключающие связи, когда подчинённая сущность связана с одной из двух родительских сущностей. Например, счёт в банке может принадлежать либо физическому лицу, либо юридическому, и не может принадлежать и тому, и другому либо не принадлежать никому.
- “you cannot get a job until you have experience, and you cannot get experience until you have a job!”

# Изменение таблицы

- Добавляем столбец

```
ALTER TABLE table_name
```

```
ADD column_name datatype
```

# Изменение таблицы

- Удаляем столбец

```
ALTER TABLE table_name
```

```
    DROP COLUMN column_name
```



# Добавление/снятие ограничений

- ALTER TABLE people  
ADD CHECK (id>0)
- ALTER TABLE people  
ADD CONSTRAINT chk\_Person CHECK (id>0 AND  
address='SPb')
- ALTER TABLE people  
DROP CONSTRAINT chk\_Person

# Изменение таблицы

- Изменяем столбец

```
ALTER TABLE table_name
```

```
ALTER COLUMN column_name datatype
```

# Изменение таблицы

- Добавляем первичный ключ

```
ALTER TABLE employee  
ADD CONSTRAINT c1 PRIMARY KEY (id)
```

id должен быть NOT NULL!

# Изменение таблицы

- Добавляем внешний ключ

```
ALTER TABLE employee  
ADD FOREIGN KEY (dno)  
REFERENCES department(dnumber)
```

# Изменение таблицы

- Добавляем ограничение

```
ALTER TABLE exd WITH NOCHECK  
ADD CONSTRAINT exd_check CHECK  
(column_a > 1)
```

# Нельзя изменять следующие столбцы

- Столбец с типом данных **timestamp**.
- Вычисляемый столбец или используемый в вычисляемом столбце.
- PRIMARY KEY или [FOREIGN KEY] REFERENCES.
- Используется в ограничениях CHECK или UNIQUE. Однако допустимо изменение длины столбца изменяемой длины, используемого в ограничении CHECK или UNIQUE.
- Некоторые изменения типов данных могут повлечь за собой изменения в данных. Например, изменение столбца типа **nchar** или типа **nvarchar** на **char** или **varchar** может вызвать преобразование символов национальных алфавитов.

# Вставка

- `INSERT INTO tbl_name (col1, col2)  
VALUES ('Character data1',1), ('Character  
data2',2), (...);`
- Будут вставлены либо все строки, либо ни одной – операция неделима!!!
- `INSERT INTO Orders (Order_ID, Customer_ID,  
Order_Date) VALUES (34, 'FRANS', '4/2/2004');`

# Вставка

- `INSERT INTO tbl_name (col1, col2)  
VALUES ('Character data',1), (...)`
- Можно:  
`INSERT INTO tbl_name (col1,col2)  
VALUES(15,col1*2);`
- Но нельзя:  
`INSERT INTO tbl_name (col1,col2)  
VALUES(col2*2,15);`



# Изменение

- UPDATE tbl\_name  
SET col1=expr1 [, col2=expr2 ...]  
[WHERE where\_definition]

```
UPDATE Product SET  
    ListPrice = ListPrice * 2  
    , MyDate=MyDate+1  
WHERE id=158;
```

# Удаление

- `DELETE FROM tbl_name  
[WHERE where_definition];`
- Если не указать условие, то будут удалены все строки таблицы
- `TRUNCATE TABLE table_name ;`

# Ограничиваем число строк

- INSERT | DELETE | UPDATE  
[ TOP (expression) [PERCENT]]

# Удаление таблицы

- DROP TABLE имя\_таблицы
- удаляет и данные, и структуру.

# Удаление

- DROP TABLE [IF EXISTS] tbl\_name