



УНИВЕРСИТЕТ ИТМО

Проектирование БД

Лекция 2. Основные цели и подходы к проектированию БД

Давайте вспомним азы (ну, на всякий)...

Отношение



Давайте вспомним азы (ну, на всякий)...

А сколько видов ключей вы знаете помните?

Давайте вспомним азы (ну, на всякий)...



Давайте вспомним азы (ну, на всякий)...

Аномалии* работы с данными

Тип аномалии	Как проявляется
Вставки	Нельзя вставить новый кортеж без нарушения требований к ключу-кандидату
Обновления	Невозможно поддержать обновление данных одним действием (требуется обновление нескольких кортежей)
Удаления	Удаление кортежа приведёт к удалению уникальных данных о какой-либо сущности

Причина возникновения – хранение в одном отношении разнородной информации

** Аномалия – противоречие между моделью предметной области и физической моделью данных*

Давайте вспомним азы (ну, на всякий)...

Сущности и связи (понадобится чуть дальше, как раз в проектировании)

Сущность	Что собой представляет
Сильная	Не зависит от какой-либо другой сущности в модели. В наборе атрибутов всегда можно выделить первичный ключ.
Слабая	Зависит от сильной сущности. Первичный ключ – составной, сформированный из первичного ключа сильной сущности и собственного ключа.
Слабая ассоциативная	Сущность, используемая в отношении «многие-ко-многим» (представляет собой дополнительную таблицу). Все связи для слабой ассоциативной сущности должны быть связями «многие».

Давайте вспомним азы (ну, на всякий)...

Сущности и связи (понадобится чуть дальше, как раз в проектировании)

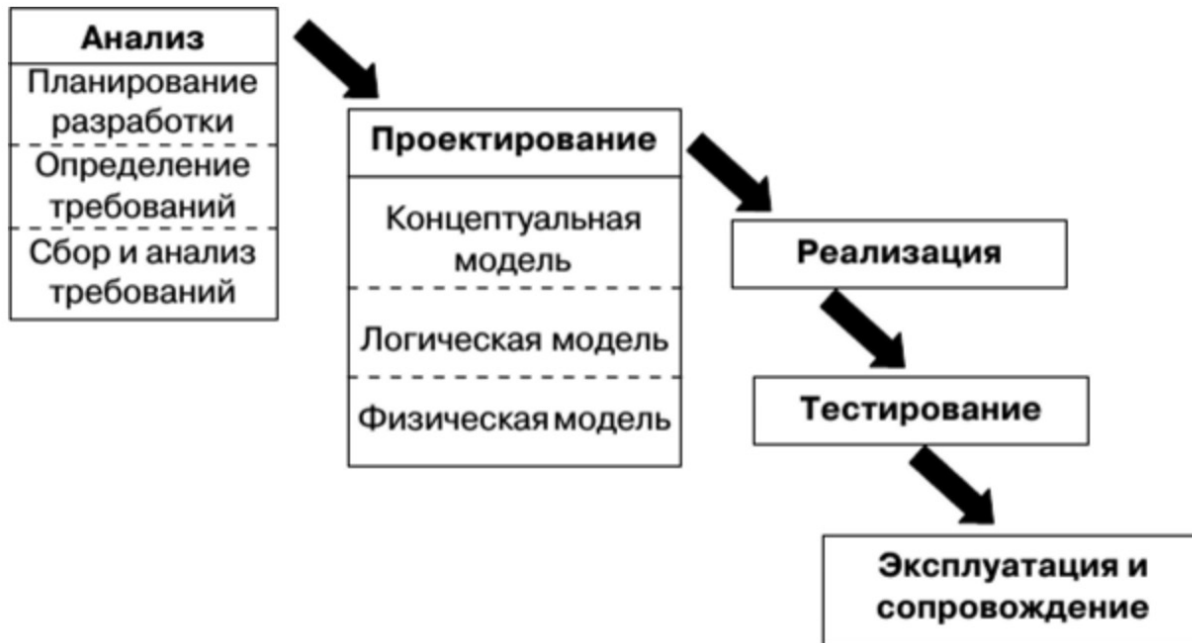
Связь	Что собой представляет
1:1	Значение атрибута A однозначно определяет значение атрибута B
1:M	Значению атрибута A могут соответствовать несколько значений атрибута B
M:1	Множеству значений атрибута A соответствует одно значение атрибута B
M:N	Множеству значений атрибута A могут соответствовать множество значений атрибута B

Давайте вспомним азы (ну, на всякий)...

Виды зависимостей ($X \rightarrow Y$) между атрибутами в отношении

Зависимость	Что собой представляет
Функциональная частичная	$R = \{ABCD\}$. AB – составной ключ. $B \rightarrow C$ Значение C можно однозначно определить по части составного ключа (B).
Функциональная полная	$R = \{ABCD\}$. $A \leftrightarrow C$ Значение C можно однозначно определить по значению A . И наоборот
Транзитивная	$R = \{ABCD\}$. $A \rightarrow B, B \rightarrow C$
Многозначная	$R = \{ABCD\}$. Одному значению A соответствует несколько значений C

...и перейдём к теме. Жизненный цикл БД



Зачем нужно проектирование?



Проектирование БД

- **Проектирование базы данных**
 - *процесс создания проекта базы данных, предназначенной для поддержки функционирования предприятия и способствующей достижению его целей*

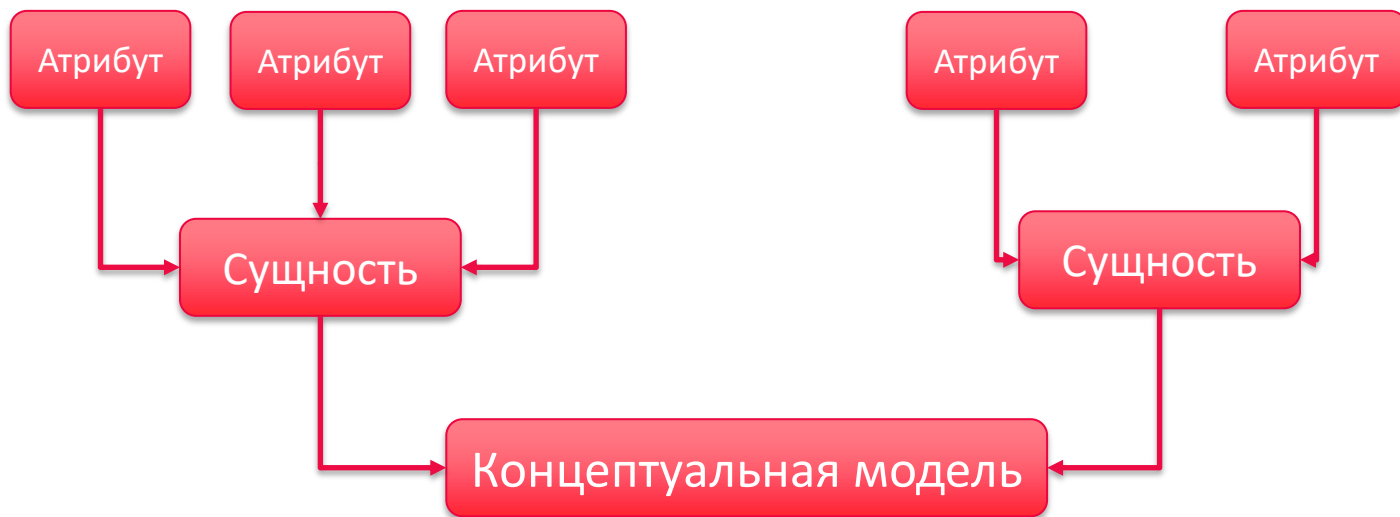
Основные цели проектирования БД

- ✓ представление данных и связей между ними
- ✓ создание модели данных
- ✓ разработка предварительного варианта проекта

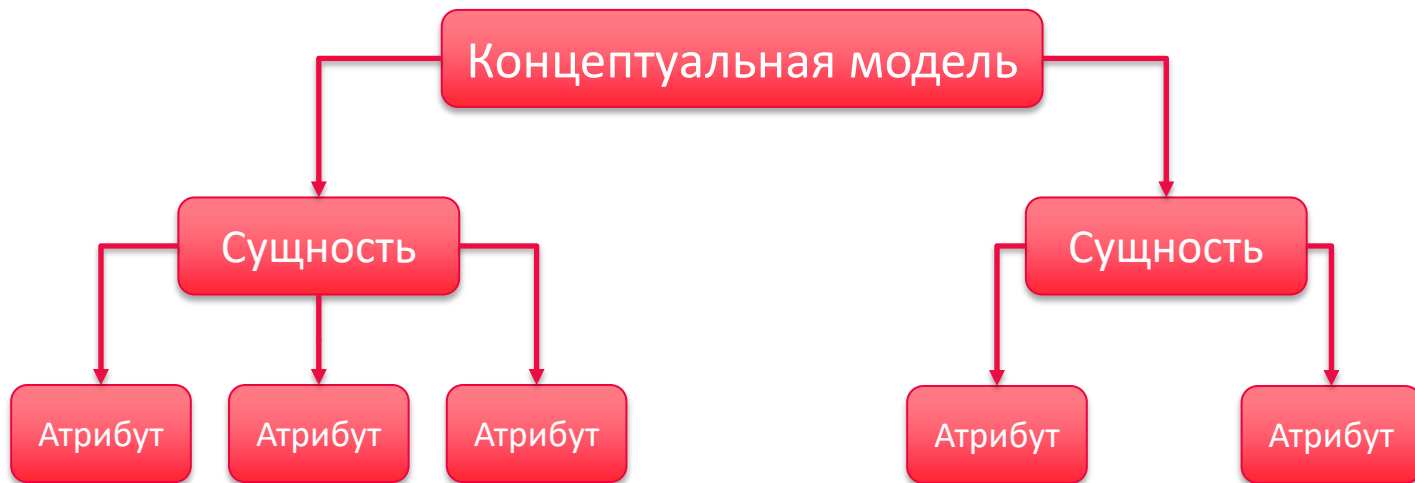
Подходы к проектированию БД

- **Восходящий**
лучше всего подходит для проектирования простых баз данных с относительно небольшим количеством атрибутов
- **Нисходящий**
для проектирования сложных баз данных
- **Изнутри наружу**
похож на восходящий подход, но отличается от него начальной идентификацией набора основных сущностей с последующим расширением круга рассматриваемых сущностей,
- **Смешанная стратегия проектирования**
сначала восходящий и нисходящий подходы используются для разных частей модели, после чего все подготовленные фрагменты собираются в единое целое.

Восходящее проектирование



Нисходящее проектирование





Метод восходящего проектирования БД



Нормализация БД

- **Избыточность данных** в ненормализованной схеме – повторение данных в БД
 - Для того чтобы полученная структура БД (ДЛМ) не обладала различными аномалиями при добавлении, обновлении или удалении данных вследствие их избыточности, необходимо осуществить проверку каждой полученной схемы отношения, как минимум, на соответствие 3НФ. Если схемы отношений не соответствуют этому условию, а они, как правило, не соответствуют, необходимо проводить процесс нормализации.
- **Нормализация БД** - это процесс проектирования в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем.

Нормализация БД

А сколько видов нормальных форм вы знаете помните?

Виды нормальных форм

Нормальная форма	Историческая справка	Краткая характеристика
Нулевая (UNF)	-	В базе данных не проводилось никакой нормализации
Первая (1NF)	1970, Кодд	Атрибуты атомарны, есть как минимум один ключ-кандидат
Вторая (2NF)	1971, Кодд	Нет частичных зависимостей (значения зависят от каждого ключа-кандидата)
Третья (3NF)	1971, Кодд	Нет транзитивных зависимостей (значения зависят только от ключей-кандидатов)
Элементарный ключ (EKNF)	1982, Заниоло	Каждая частичная зависимость включает в себя либо супер-ключ, либо подраздел элементарного ключа

Виды нормальных форм

Нормальная форма	Историческая справка	Краткая характеристика
Бойса-Кодда (BCNF)	1974, Кодд	Нет избыточности от какой-либо функциональной зависимости
Четвёртая (4NF)	1977, Фагин	У каждой частичной многозначной зависимости есть супер-ключ
Основного кортежа (ETNF)	2012,	Компонентом каждой явной зависимости является супер-ключ
Пятая (5NF)	1979, Фагин	Каждая частичная зависимость соединения подразумевается ключом-кандидатом
Доменно-ключевая (DKNF)	1981, Фагин	Каждое ограничение является следствием ограничений домена и ключевых ограничений
Шестая (6NF)	2002, Дэйт	Каждая зависимость присоединений тривиальна

Если наглядно

	UNF	1NF	2NF	3NF	EKNF	BCNF	4NF	ETNF	5NF	DKNF	6NF
Первичный ключ (без дубликатов)											
Нет повторяющихся групп											
Атомарные атрибуты (одно значение, без перечислений)											
Нет частичных зависимостей (значения зависят от каждого ключа-кандидата)											
Нет транзитивных зависимостей (значения зависят только от ключей-кандидатов)											
Каждая частичная ф. з. включает в себя либо супер-ключ, либо подраздел элементарного ключа											
Нет избыточности от какой-либо функциональной зависимости											
У каждой частичной многозначной зависимости есть супер-ключ											
Компонентом каждой явной зависимости соединения является супер-ключ											
Каждая нетривиальная зависимость соединения подразумевается ключом-кандидатом											
Каждое ограничение является следствием ограничений домена и ключевых ограничений											
Каждая зависимость присоединения тривиальна											

Процесс нормализации

Удаление
повторяющихся
групп(->1НФ)

Удаление
частичных
функциональных
зависимостей(-
>2НФ)

Удаление
транзитивных
зависимостей(-
>3НФ)

Удаление из ФЗ
оставшиеся
аномалий(-
>НФБК)

Удаление
нетривиальных
зависимостей(-
>4НФ)

Удаление
зависимостей
соединения(-
>5НФ)



Если по коду (UNF → 1NF)

```
create table seasonLineUp (  
  team varchar (50),  
  engine varchar (50),  
  driver varchar (50)  
);  
insert into seasonLineUp values  
  ('Mercedes', 'Mercedes',  
   'Lewis Hamilton, George Russell'),  
  ('Aston Martin', 'Mercedes',  
   'Fernando Alonso, Lance Stroll'),  
  ('Alpine', 'Renault',  
   'Esteban Okon, Pierre Gasly');
```

```
create table seasonLineUp (  
  team varchar (50),  
  engine varchar (50),  
  driver varchar (50)  
);  
insert into seasonLineUp values  
  ('Mercedes', 'Mercedes', 'Lewis Hamilton'),  
  ('Mercedes', 'Mercedes', 'George Russell'),  
  ('Aston Martin', 'Mercedes', 'Fernando Alonso'),  
  ('Aston Martin', 'Mercedes', 'Lance S troll'),  
  ('Alpine', 'Renault', 'Esteban Okon');  
  ('Alpine', 'Renault', 'Pierre Gasly');
```

Все столбцы содержат атомарные неделимые значения.

Нет повторяющихся групп или массивов.

Каждая строка уникальна.

Если по коду (1NF → 2NF)

```
create table seasonLineUp (  
    team varchar (50),  
    driver varchar (50),  
    status varchar (1),  
    engine varchar (50),  
    primary key (team, driver)  
);  
insert into seasonLineUp values  
    ('Mercedes', 'Lewis Hamilton', 'E', 'Mercedes'),  
    ('Mercedes', 'George Russell', 'E', 'Mercedes'),  
    ('Red Bull', 'Max Verstappen', '1', 'Honda'),  
    ('Red Bull', 'Sergio Peres', '2', 'Honda'),  
    ('Williams', 'Alex Albon', '1', 'Mercedes'),  
    ('Williams', 'Logan Sargeant', '2', 'Mercedes')  
;
```

```
create table seasonLineUp (  
    id integer  
    team varchar (50),  
    driver varchar (50),  
    status varchar (1),  
    engine varchar (50),  
    primary key (team, driver)  
);  
insert into seasonLineUp values  
    (1, 'Mercedes', 'Lewis Hamilton', 'E', 'Mercedes'),  
    (2, 'Mercedes', 'George Russell', 'E', 'Mercedes'),  
    (3, 'Red Bull', 'Max Verstappen', '1', 'Honda'),  
    (4, 'Red Bull', 'Sergio Peres', '2', 'Honda'),  
    (5, 'Williams', 'Alex Albon', '1', 'Mercedes'),  
    (6, 'Williams', 'Logan Sargeant', '2', 'Mercedes')  
;
```




Если по коду (1NF → 2NF)

```
create table seasonLineUp (  
    team varchar (50),  
    driver varchar (50),  
    status varchar (1),  
    engine varchar (50),  
    primary key (team, driver)  
);  
insert into seasonLineUp values  
    ('Mercedes', 'Lewis Hamilton', 'E', 'Mercedes'),  
    ('Mercedes', 'George Russell', 'E', 'Mercedes'),  
    ('Red Bull', 'Max Verstappen', '1', 'Honda'),  
    ('Red Bull', 'Sergio Peres', '2', 'Honda'),  
    ('Williams', 'Alex Albon', '1', 'Mercedes'),  
    ('Williams', 'Logan Sargeant', '2', 'Mercedes')  
;
```

```
select t1.*, t2.engine from  
seasonLineUp t1, teams t2  
where t2.team = t1.team;
```

```
create table seasonLineUp (  
    team varchar (50),  
    driver varchar (50),  
    status varchar (1),  
    primary key (team, driver)  
);  
insert into seasonLineUp values  
    ('Mercedes', 'Lewis Hamilton', 'E'),  
    ('Mercedes', 'George Russell', 'E'),  
    ('Red Bull', 'Max Verstappen', '1'),  
    ('Red Bull', 'Sergio Peres', '2'),  
    ('Williams', 'Alex Albon', '1'),  
    ('Williams', 'Logan Sargeant', '2');  
  
create table teams (  
    team varchar (50),  
    engine varchar (50)  
);  
insert into teams values  
    ('Mercedes', 'Mercedes'),  
    ('Red Bull', 'Honda'),  
    ('Williams', 'Mercedes');
```



Если по коду (2NF → 3NF)

```
create table driverDetails (  
    driver varchar (50),  
    date_of_birth date,  
    age integer -- ! возраст может быть  
                вычислен на основе даты рождения,  
                поэтому это транзитивная зависимость  
);  
  
insert into driverDetails values  
('Lewis Hamilton', '1985-01-07', 39),  
('Max Verstappen', '1997-09-30', 26)  
;
```

```
create table driverDetails (  
    driver varchar (50),  
    date_of_birth date  
);  
insert into driverDetails values  
('Lewis Hamilton', '1985-01-07'),  
('Max Verstappen', '1997-09-30')  
;
```

```
select driver,  
       date_of_birth,  
       date_part ('year',  
                 age(date_of_birth::date)) as age  
from driverDetails;
```



Если по коду (2NF → 3NF)

```
create table driversLineUp (  
    driver varchar (50),  
    team varchar (50),  
    engine varchar (50)  
);  
  
insert into driversLineUp values  
    ('Lewis Hamilton', 'Mercedes', 'Mercedes'),  
    ('Max Verstappen', 'Red Bull', 'Honda'),  
    ('Fernando Alonso', 'Aston Martin', 'Mercedes');
```

```
select t1.*, t2.engine  
    from driversLineUp t1, teams t2  
    where t2.team = t1.team;
```

```
create table driversLineUp (  
    driver varchar (50),  
    team varchar (50)  
);  
  
insert into driversLineUp values  
    ('Lewis Hamilton', 'Mercedes'),  
    ('Max Verstappen', 'Red Bull'),  
    ('Fernando Alonso', 'Aston Martin');  
  
create table teams (  
    team varchar (50),  
    engine varchar (50)  
);  
  
insert into teams values  
    ('Mercedes', 'Mercedes'),  
    ('Red Bull', 'Honda'),  
    ('Williams', 'Mercedes');
```



Если по коду (3NF → ECNF)

```
create table teamStaff (  
  team varchar (50),  
  managerId integer,  
  managerName varchar (50),  
  primary key (team, managerId)  
);
```

```
create table teamManagers (  
  managerId integer,  
  managerName varchar (50),  
  primary key (managerId)  
);  
  
create table teamStaff (  
  team varchar (50),  
  managerId integer,  
  primary key (team),  
  foreign key (managerId) references teamManagers  
  (managerId)  
);
```

В этой таблице **managerName** функционально зависит от **managerId**, который сам по себе не является супер-ключом, но входит в альтернативный ключ {team, managerName}

Если по коду (3NF → BCNF)

```
create table tracks (  
    trackName varchar (50),  
    trackType varchar (50),  
    trackPlace varchar (50),  
    country varchar (50),  
    primary key (trackName, trackType)  
);
```

```
create table trackTypes (  
    trackTypeId integer,  
    trackTypeName varchar (50),  
    trackPlace varchar (50),  
    primary key (trackTypeId)  
);  
  
create table tracks (  
    trackName varchar (50),  
    trackTypeId integer,  
    country varchar (50),  
    primary key (trackName, trackTypeId),  
    foreign key (trackTypeId) references  
    trackTypes(trackTypeId)  
);
```

BCNF устраняет зависимость ключевого атрибута **trackType** от неключевого trackPlace путем разделения **trackType** на собственную таблицу.



Если по коду (BCNF → 4NF)

```
create table driverTestRoutines (  
  driver varchar (50),  
  testDay varchar (50),  
  testTeam varchar (50)  
);
```

4NF предназначен для обработки более сложных сценариев с многозначными зависимостями, когда один ключ может привести к нескольким независимым значениям в другом столбце.

```
create table testDays (  
  dayId integer,  
  testDayName varchar (50),  
  primary key (dayId)  
);  
  
create table testTeams (  
  teamId integer,  
  teamName varchar (50),  
  primary key (teamId)  
);  
  
create table driverTestRoutines (  
  driver varchar (50),  
  dayId integer,  
  teamId integer,  
  primary key (driver),  
  foreign key (dayId) references testDays (dayId),  
  foreign key (teamId) references testTeams (teamId)  
);
```





Если по коду (ETNF)

```
create table marketingEvents (  
    eventId integer,  
    eventName varchar (50),  
    primary key (eventId)  
);  
create table drivers (  
    driverId integer,  
    driverName varchar (50),  
    primary key (driverId)  
);  
create table driverMarketingSchedule (  
    driverId integer,  
    eventId integer,  
    dateOfEvent date,  
    foreign key (driverId) references drivers (driverId),  
    foreign key (eventId) references marketingEvents (eventId)  
);
```




Если по коду (5NF)

```
create table testSpecs (  
    testSpecId integer primary key,  
    testSpecName varchar (50)  
);  
create table testTeams (  
    testTeamId integer primary key,  
    testTeamManager varchar (50)  
);  
create table drivers (  
    driverId integer primary key,  
    driverName varchar (50)  
);  
create table testingRoutines (  
    driverId integer,  
    testSpecId integer,  
    testTeamId integer,  
    foreign key (driverId) references drivers (driverId),  
    foreign key (testSpecId) references testSpecs (testSpecId),  
    foreign key (testTeamId) references testTeams (testTeamId)  
);
```

Если по коду (DKNF)

DKNF — это скорее теоретическая конструкция, где таблица находится в DKNF, если каждое ее ограничение является логическим следствием определения ключей и доменов.

Трудно привести конкретный пример DKNF, поскольку это скорее теоретический идеал. На практике, если таблица находится в 5NF и все ограничения (например, проверочные ограничения, ограничения внешнего ключа) основаны на определениях ключей и ограничениях домена, ее можно считать находящейся в DKNF.

Если по коду (6NF)

```
create table drivers (  
    driverId integer primary key,  
    driverName varchar (50)  
);  
  
create table driverStatus (  
    driverId integer,  
    statusName varchar (50),  
    effectiveDate date,  
    foreign key (driverId) references drivers (driverId)  
);  
  
create table driverSalary (  
    driverId integer,  
    salary integer,  
    effectiveDate date,  
    foreign key (driverId) references drivers (driverId)  
);
```

Вместо резюме

1NF – атомарность или 1 поле содержит 1 значение.

2NF – каждой таблице свой уникальный ключ.

3NF – 1 сущность = 1 таблица

4NF – каждую связь M:N (многие ко многим) разбить на многие к одному.

Применение NF высшего порядка может быть целесообразно в зависимости от формируемой модели предметной области

Минусы нормализации

Моделирование структуры базы данных при помощи алгоритма нормализации имеет серьезный недостаток:

методика нормализации предполагает первоначальное размещение всех атрибутов проектируемой предметной области в одном отношении, что является *логически очень неестественной операцией*

Минусы нормализации (это ещё не всё)

- Интуитивно разработчик сразу проектирует несколько отношений в соответствии с обнаруженными сущностями. Даже если совершить насилие над собой и создать одно или несколько отношений, включив в них все предполагаемые атрибуты, то совершенно неясен смысл полученного отношения.
- Невозможно сразу определить полный список атрибутов. Пользователи имеют привычку называть разными именами одни и те же вещи или наоборот, называть одними именами разные вещи. Для проведения процедуры нормализации необходимо выделить зависимости атрибутов, что тоже очень нелегко.

А ведь ещё есть...

Денормализация!

Это обратный процесс, когда приходится объединять данные в одну таблицу или добавлять дополнительные поля. Такие действия сразу же ведут к появлению избыточности данных, но уменьшают затраты на получение информации.

ВАЖНО! Денормализация используется НЕ вместо нормализации, а ПОСЛЕ нее. Здесь нужно найти золотую середину: до каких пор мы можем разбивать данные, чтобы облегчить выполнение команд `insert`, `update` и `delete`, и когда можно допустить некоторую избыточность для ускорения выполнения команды `select`.

Денормализация vs Нормализация

	Денормализация	Нормализация
+	<ul style="list-style-type: none"> ➤ Более быстрое чтение данных ➤ Более простые запросы выгодны разработчикам ➤ Меньше вычислений при операциях чтения 	<ul style="list-style-type: none"> ➤ Быстрее запись ➤ Нет избыточности данных ➤ Ниже сложность базы данных ➤ Данные всегда согласованы
-	<ul style="list-style-type: none"> ➤ Медленнее запись ➤ Выше сложность базы данных ➤ Вероятность несогласованности данных ➤ Нужно больше СХД и оперативной памяти 	<ul style="list-style-type: none"> ➤ Медленнее чтение ➤ Тяжелые запросы могут привести к перегрузке и сбою оборудования ➤ Требуется join'ы, т.к. данные не дублируются ➤ Индексация из-за join'ов не так эффективна

В общем...

...нормализуем, пока
не станет больно.

...денормализуем,
пока оно работает.



А зачем же она тогда вообще нужна?

В реальном проектировании структуры базы данных применяются другой метод - так называемое семантическое или нисходящее моделирование, опирающееся на смысл моделируемых данных. В качестве инструмента семантического моделирования используются различные варианты диаграмм "сущность-связь (ER)" с построением концептуальной модели базы данных. Обычно проектирование с помощью ER-диаграмм позволяет построить БД, нормализованную до 3НФ или НФБК.

Но нормализация не является полностью бесполезной. С её помощью удобно логически проверять созданные ER-модели.

Нисходящее проектирование БД

Анализ предметной
области

Разработка
информационно—
логической модели
предметной области

Формирование
даталогической
модели БД

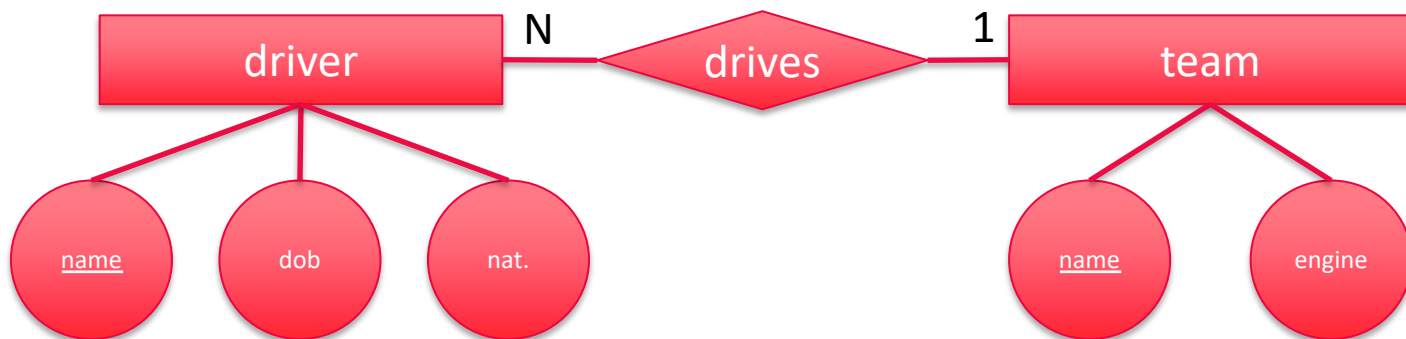
Незначительная
нормализация

Формирование
физической модели
БД

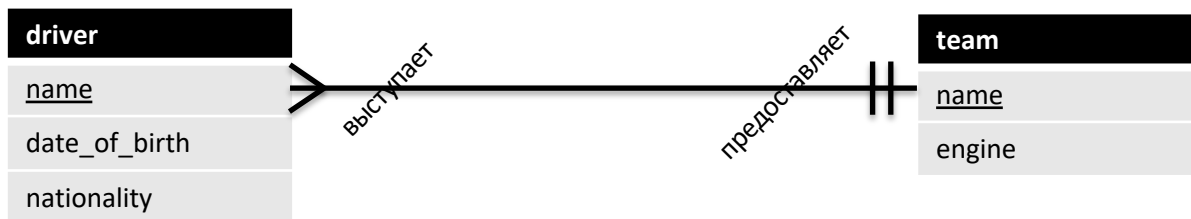
Сферы концептуального подхода

- реальный мир или объектная система
- информационная сфера
- даталогическая сфера.

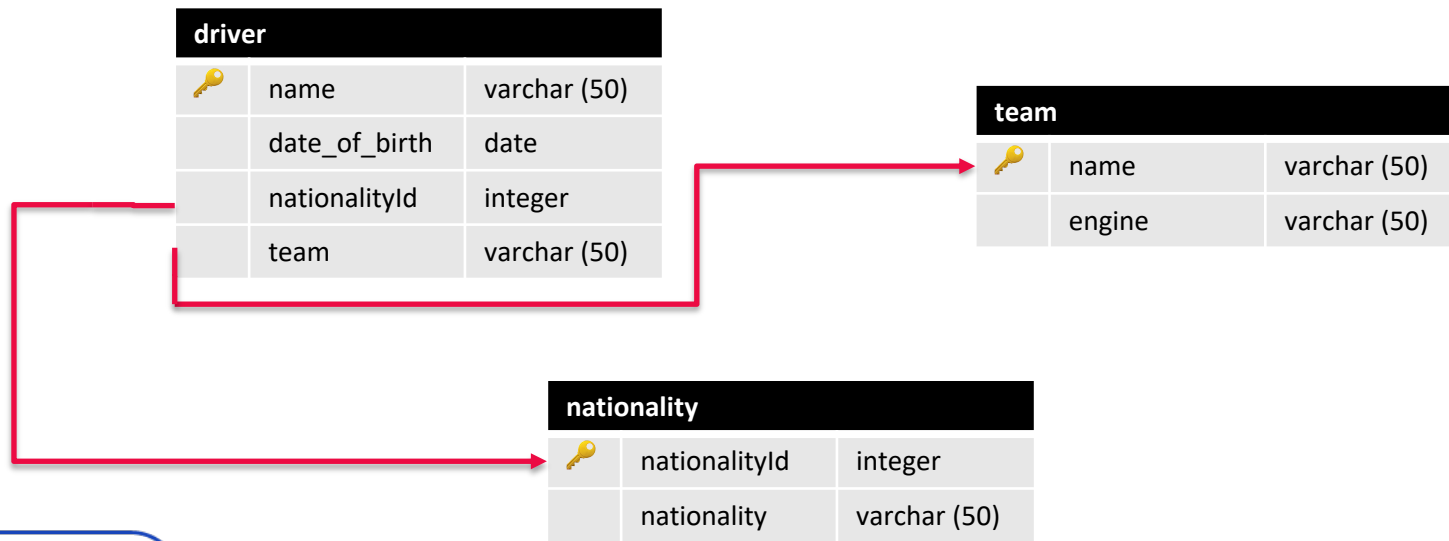
Инфологическая модель



ИЛИ



Даталогическая модель



Особенности концептуального подхода

- на втором этапе проектирования ИЛМ можно преобразовать в любую даталогическую модель – иерархическую, сетевую, реляционную, noSql.
- достаточно формализован и используется в CASE (Computer Aided System/Software Engineering — компьютерное проектирование программного обеспечения и систем) средствах
 - особенно эффективно их использование при создании крупных корпоративных ИС с большим коллективом разработчиков;
 - современные CASE – средства позволяют поддерживать как начальные этапы разработки ИС, так и проектирование, и генерацию баз данных и пользовательских интерфейсов;
 - CASE – средства обеспечивают качество принимаемых технических решений и подготовку проектной документации;

Сравнение методов проектирования

Критерии	«Нисходящее» проектирование (Концептуальный подход)	«Восходящее» проектирование (Метод нормализации)
Степень описания семантики (смысла) предметной области	Высокая	Низкая (начальная модель – ДЛМ РБД, термины РМД не предусматривают возможность описания полного смысла предметной области.)
Вероятность появления ошибок в последующей работе ИС	Низкая (при условии качественного проектирования)	Высокая
Степень формализации процесса (возможность автоматизации процесса)	Высокая	Отсутствует
Объем трудозатрат при приведении ДЛМ БД к заданной НФ	Небольшой	Очень большой

MongoDB

- Базовыми понятиями в MongoDB являются коллекции и документы.
- Коллекция является эквивалентом таблицы в реляционных системах управления базами данных.
- Документ – запись в коллекции и базовая единица данных в MongoDB
- Документы в коллекции могут иметь разные поля

Правила перехода к документно-ориентированной модели базы данных

- Если степень связи 1:1 и класс принадлежности обоих сущностей является обязательным, то создается один документ, одним из полей которого является вложенный документ, содержащий данные о второй сущности
- Если степень связи 1:1 и класс принадлежности одной сущности является обязательным, а другой – необязательным, то создается один документ, одним из полей которого может быть вложенный документ, содержащий данные о второй сущности

Правила перехода к документно-ориентированной модели базы данных

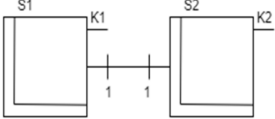

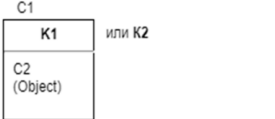
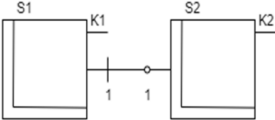
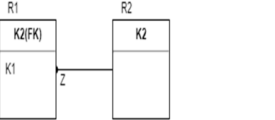
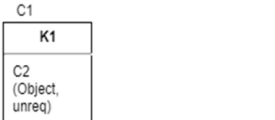
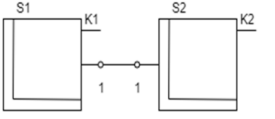
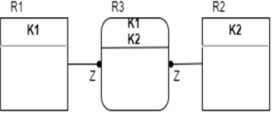
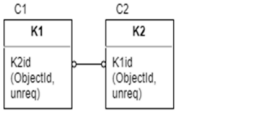
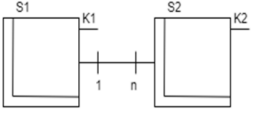
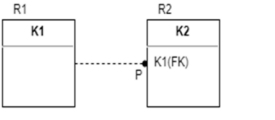
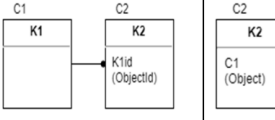
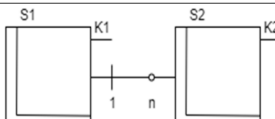
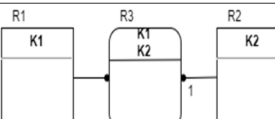
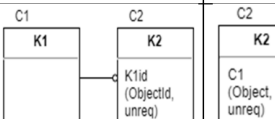
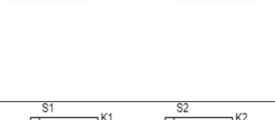
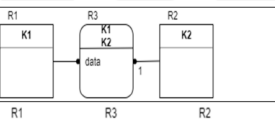
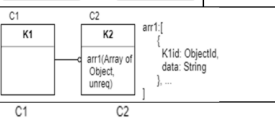
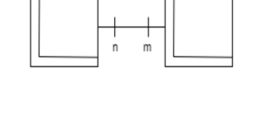
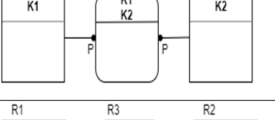
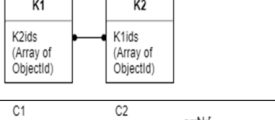
- Если степень связи 1:1 и класс принадлежности ни одной сущности не является обязательным, то создается два документа, при этом в каждом из них может содержаться поле, хранящее ссылку(идентификатор) на другой документ.
- Если степень связи 1:n и класс принадлежности n-связной сущности является обязательным, возможны 2 варианта: -
 - один документ, одним из полей которого является вложенный документ, содержащий данные о второй сущности
 - два документа, при этом второй документ содержит поле, хранящее ссылку(идентификатор) на первый документ.

Правила перехода к документно-ориентированной модели базы данных

- Если степень связи 1:n, класс принадлежности n-связной сущности является необязательным и нет дополнительных полей, то возможны 2 варианта:
 - один документ, одним из полей которого может быть вложенный документ, содержащий данные о второй сущности
 - два документа, при этом второй документ может содержать поле, хранящее ссылку(идентификатор) на первый документ.

Правила перехода к документно-ориентированной модели базы данных

- Если степень связи $m:n$ и нет дополнительных полей, создается 2 документа, каждый из которых содержит массив ссылок на другой документ.
- Если степень связи $m:n$ и есть дополнительные поля, создается 2 документа, каждый из которых содержит массив объектов следующего типа: `arrN:[{ KNid: ObjectId, data: String }, ...]`, где `KNid` – ссылка на другой документ `data` – дополнительное поле

Концептуальное проектирование S - сущность	Реляционное проектирование (методология IDEF1X) R - отношение	Документно-ориентированное проектирование C - коллекция
		
		
		
		
		
		
		

Спасибо за внимание!

www.ifmo.ru

IT'sMO *re than a*
UNIVERSITY