# Модели данных

Модель данных описывает некоторый набор родовых понятий и признаков, которыми должны обладать все конкретные СУБД и управляемые ими базы данных, основанные на этой модели.

По К. Дейту для любой модели данных принято выделять три части:

1. Структурная часть – описывает основные логические структуры данных, которые могут применяться на уровне пользователя при организации БД
2. Манипуляционная часть – обеспечивает модельный язык БД
3. Целостная часть – специфицирует механизмы ограничений целостности, которые обязательно должны поддерживаться во всех реализациях СУБД

Изученные модели данных:

- Дореляционные (60-е – 70-е г.г.):
  - Иерархическая – деревья
  - Сетевая – произвольные ориентированные графы
  - Инвентированные списки – таблицы с физической адресацией записей
- Реляционная (Э. Кодд, 1969 г.)
- Модель данных SQL

# Проблема «потери соответствия»

К началу 90-х годов SQL-ориентированные СУБД заняли на мировом рынке доминирующее положение, которое удерживают и по настоящий момент.

В 80-е годы стала приобретать популярность парадигма объектно-ориентированного программирования (языки Smalltalk, Ada, Objective-C, C++). В 90-е годы объектно-ориентированное программирование вошло в моду и стало завоевывать преимущественное положение в программной индустрии. Б*о*льшая часть современных производственных приложений, связанных с потребностью в обработке больших объемов данных, разрабатывается на различных объектно-ориентированных языках (C++, Java, C#).

В связи с этим возникла проблема «потери соответствия» (impedance mismatch) между системами типов и средствами доступа к данным объектно-ориентированных языков программирования и SQL-ориентированных СУБД. Это затрудняет разработку приложений, которые по своей специфике вынуждены часто обращаться к базе данных для доступа к требуемым им данным.

# Предложенные подходы

1989 г., М. Аткинсон и др., «Манифест систем объектно-ориентированных баз данных»:

- СУБД должна иметь возможность хранить данные произвольно сложной структуры и обеспечивать развитую систему типов данных
- Реляционная модель данных и язык SQL являются ограниченными и не подходят для этих целей
- Предлагается использовать в СУБД объектно-ориентированную модель и формулируются базовые требования к ООСУБД

1990 г., М. Стоунбрейкер и др., «Манифест систем баз данных третьего поколения»:

- СУБД должна иметь возможность хранить данные произвольно сложной структуры и обеспечивать развитую систему типов данных
- Можно добиться требуемых результатов, эволюционно развивая SQL

1995 г., К. Дейт, Х. Дарвен, «Третий манифест»:

- Идеи первых двух манифестов необоснованы и плохо проработаны
- Классическая реляционная модель данных необходима и достаточна для использования в современных СУБД
- Фактически предлагается новая, «истинная реляционная» модель

# Объектно-ориентированная модель

Литеральные типы данных – могут использоваться в качестве типов атрибутов внутри объектов. Значения литеральных типов не могут самостоятельно храниться в базе данных.

Атомарные литеральные типы: числовые, символьные, булевские, темпоральные (принятые как в традиционных языках программирования, так и в языках баз данных).

Конструируемые литеральные типы:

- Запись (аналог структуры в языках C/C++)
- Коллекции:
  - Множество (неупорядоченная, без дубликатов)
  - Мультимножество (неупорядоченная, с дубликатами)
  - Массив (упорядоченная, с возможностью доступа к элементу по индексу)
  - Список (упорядоченная, с возможностью быстрой вставки и удаления элементов)
  - Словарь (множество пар <ключ, значение>)

# Объектные типы данных

Объектные типы данных: атомарные (классы) и объектные коллекции.
Каждый объектный тип имеет операцию создания и инициализации (конструктор).
Значения объектных типов имеют объектный идентификатор (OID) и могут самостоятельно храниться в базе данных.
OID автоматически генерируется СУБД, является уникальным во всей БД, не зависит от состояния объекта (значений его атрибутов).
При определении класса указывается его внутренняя структура (набор атрибутов и связей) и набор операций. Одиночное наследование поддерживается в обязательном порядке, поддержка множественного наследования является желательной. Поддерживаются только бинарные связи.
В отличие от литеральных коллекций, объектные коллекции могут самостоятельно храниться в БД, набор операций для каждого из типов коллекций предопределен.
Экстентом объектного типа называется именованная коллекция типа множества, элементами которой являются объекты данного типа.

# Виды эквивалентности объектов

В связи с наличием у объектов уникальных идентификаторов различают три вида эквивалентности объектов:

1. Эквивалентность по идентификации (идентичность): объекты являются идентичными тогда и только тогда, когда их идентификаторы совпадают.
2. Поверхностная эквивалентность: два объекта поверхностно эквивалентны тогда и только тогда, когда их атрибуты идентичны (выполняется поверхностное сравнение: простые атрибуты сравниваются по значению, а объектные – по идентификатору).
3. Эквивалентность по значению (равенство): два объекта равны тогда и только тогда, когда значения всех их атрибутов одинаковы (выполняется глубокое сравнение: при наличии атрибутов объектного типа должны быть одинаковыми значения всех их атрибутов рекурсивно).

Очевидно, что два идентичных объекта будут также равны и поверхностно эквивалентны, в то время как обратное неверно. Аналогично, поверхностная эквивалентность объектов означает их равенство, но не наоборот.

# Язык OQL

OQL (Object Query Language) – базовое средство манипулирования в объектно-ориентированной модели.

• OQL не является вычислительно полным языком. Он представляет собой простой язык запросов (обеспечивается декларативный доступ к объектам).

• OQL очень близок по синтаксису к SQL/92. Расширения относятся к объектно-ориентированным понятиям, таким как объекты, объектные идентификаторы, путевые выражения (обход по связям), полиморфизм, вызов операций, и проч.

• В OQL поддерживаются средства доступа ко всем возможным структурам данных, допускаемых в структурной части модели.

• OQL является функциональным языком, допускающим неограниченную композицию операций, если операнды не выходят на пределы системы типов (результат любого запроса обладает типом, принадлежащим к объектной модели, и поэтому к результату запроса может быть применен новый запрос).

• Операторы языка OQL могут вызываться из любого языка программирования, для которого определены правила связывания. И, наоборот, в запросах OQL могут присутствовать вызовы операций, запрограммированных на этих языках.

• В OQL не определяются явные операции обновления, а используются вызовы операций, определенных в объектах для целей обновления.

# Язык OQL

Результатом допустимых в OQL выражений запросов могут являться:

- индивидуальный объект;
- коллекция объектов;
- коллекция литеральных значений;
- индивидуальное литеральное значение.

Пример: определить руководителей отделов и тех служащих их отделов, зарплата которых превышает 20000 руб.

```
SELECT DISTINCT STRUCT ( DEPT_MNG: D.DEPT_MANAGER,
                                    EMP: ( SELECT E FROM D.CONSISTS_OF AS E WHERE
E.EMP_SALARY > 20000.00 )
            ) FROM DEPARTMENTS D;
```

DEPARTMENTS – имя экстента объектного типа DEPARTMENT

CONSISTS_OF – имя роли (конца связи) типа EMPLOYEE в связи с типом DEPARTMENT

Результат запроса имеет тип set <struct { OID <EMPLOYEE> DEPT_MNG; bag < EMPLOYEE> EMP }>

# Прочие языковые средства и поддержка целостности данных

Язык ODL (Object Definition Language) – позволяет описывать схему БД в виде набора интерфейсов объектных типов (описание атрибутов, связей между объектными типами, а также операций и их параметров). ODL не является языком программирования, требуются средства отображения конструкций ODL в объектно-ориентированные языки программирования для реализации типов. В качестве языка манипулирования объектами (создание, модификация, удаление) предполагается использование объектно-ориентированных языков программирования. С этой целью языки программирования расширяются дополнительными конструкциями либо библиотечными элементами. Правила связывания в курсе лекций не рассматриваются.

В объектной модели отсутствует аналог проверочного ограничения целостности реляционной модели данных.
Ссылочная целостность поддерживается для связей «один-ко-многим». В этом случае объекты на стороне связи «один» рассматриваются как предки, а объекты на стороне связи «многие» – как потомки, и ООСУБД обязана следить за тем, чтобы не образовывались потомки без предков.

# Подходы к созданию и удалению объектов

Различают перманентные (persistent) и транзитные (transient) объекты. Первые реально сохраняются в БД, т.е. приобретают свойство хранимости (устойчивости). Вторые существуют временно в течение сеанса работы приложения.
Существуют следующие подходы к созданию (конструированию) объектов:

1. Создание экземпляра (путем вызова его конструктора) приводит к его добавлению в базу данных (автоматически предполагается устойчивость экземпляров).
2. Вызов конструктора предполагает создание транзитного экземпляра. Тогда созданный во время работы программы объект уничтожается при ее завершении, если он не сделан устойчивым путем вызова специального метода.
3. Возможно конструировать как перманентные, так и транзитные экземпляры. Тип создаваемого экземпляра определяется специальным параметром конструктора.

Удаление объектов может осуществляться двумя альтернативными способами:

1. Объект физически удаляется в любом случае. Контроль ссылочной целостности на уровне СУБД при этом не производится и полностью возлагается на приложение.
2. Объект имеет счетчик ссылок на него, при ненулевом значении этого счетчика вместо уничтожения объекта выставляется "флаг уничтожения" (tombstone), и объект удаляется только после того, как счетчик обнулится.

# Объектно-ориентированные СУБД

Известные реализации: db4o, Objectivity/DB, Versant

- Ни одна из СУБД не поддерживает ограничения целостности, за исключением ссылочной, да и то в ограниченном объеме, поддержка целостности данных фактически является ответственностью приложения
- Неустановленные значения поддерживаются только для объектных ссылок, для других типов – это забота приложения
- Механизмы триггеров, хранимых процедур и т.п., не поддерживаются
- Ограничение доступа к данным со стороны различных пользователей не поддерживается или ограниченно поддерживается в Versant
- Ни ODL, ни OQL не поддерживаются (каждая ООСУБД предлагает собственные языковые средства), запросы, результатами которых являются литеральные значения или их множества не поддерживаются
- db4o поддерживает только Java или C#, причем для этих языков выпускаются два разных продукта, остальные хотя и декларируют независимость от языков, эта независимость является условной (необходимо соблюсти ряд требований, например, по использованию определенных типов данных)
- Перевод существующего приложения достаточно трудоемок (например, необходимо перейти на типы данных, поддерживаемые той или иной ООСУБД)

# Современное состояние

С конца 90-х годов в области ООСУБД наблюдался явный упадок. Основные причины:

- Недостатки ООСУБД (предыдущий слайд), главным из которых является зависимость от языков программирования
- Отсутствие крупных игроков на рынке (таких как Oracle и IBM)
- Появление объектно-реляционных возможностей в большинстве популярных SQL-ориентированных СУБД
- Маркетинговая политика крупных компаний, которые сумели внушить пользователям должное доверие к универсальности, надежности и масштабируемости своих SQL-ориентированных решений

С середины 2000-х некоторое повышение интереса к ООСУБД:
2005 г. – возникновение концорциума ODBMS.ORG
2006 г. – сформирована группа (в рамках OMG) по подготовке новой версии стандарта объектно-ориентированной модели данных
2008 г. – первая после длительного перерыва конференция, посвященная проблемам ООСУБД

# Объектно-реляционные расширения SQL

Типы данных, введенные в SQL:1999 и не относящиеся к ОО расширениям:

- Массивы: datatype ARRAY [max_cardinality]
- Мультимножества: datatype MULTISET
- Анонимные строчные типы: ROW( field1, …, fieldN), где field ::= name, type, options

Типы данных, определяемые пользователем:

- Индивидуальные типы: CREATE TYPE UDT_name AS base_type_name FINAL;
- Структурные типы: CREATE TYPE UDT_name [UNDER UDT_name] AS (attribute_list) [ INSTANTIABLE | NOT INSTANTIABLE ] { FINAL | NOT FINAL } [reference_specification] [method_specification_list]

Индивидуальный тип – именованный тип данных, основанный на единственном предопределенном типе. Индивидуальный тип не наследует от своего опорного типа набор операций над значениями.

Структурный тип данных – именованный типы данных, включающий один или более атрибутов любого из допустимых в SQL типа данных. Дополнительные механизмы: наследование, определяемые пользователями методы.

# Типизированные таблицы

CREATE TABLE typed_table_name OF UDT_name [UNDER typed_table_name] [(typed_table_element_list)]
typed_table_element ::= constraint_definition | self-ref_column_definition | column_options
CREATE VIEW typed_view_name OF UDT_name UNDER base_table_name AS query_expression
При определении типизированной таблицы указывается ранее определенный структурный тип, и если в нем содержится n атрибутов, то в таблице образуется n+1 столбец, дополнительный (первый) столбец называется самоссылающимся и содержит типизированные уникальные идентификаторы строк, которые могут генерироваться системой при вставке строк в типизированную таблицу, явно указываться пользователями или состоять из комбинации значений других столбцов. Можно определить подтаблицу типизированной таблицы, если структурный тип подтаблицы является непосредственным подтипом структурного типа супертаблицы.
В случае типизированных представлений выражение запроса должно основываться на единственной базисной таблице или представлении (базисная таблица и определяемое представление должны быть ассоциированы с одним и тем же структурным типом).

# Механизмы генерации ссылочных значений

| Спецификация ссылки в UDT | Определение самоссылающегося столбца |
|---|---|
| REF IS SYSTEM GENERATED | REF IS column_name SYSTEM GENERATED |
| REF USING predefined_type | REF IS column_name USER GENERATED |
| REF USING (attribute_list) | REF IS DERIVED |

Определение типа атрибута – ссылки на другой структурный тип:
REF( UDT_name ) [SCOPE typed_table_name REFERENCES ARE [NOT] CHECKED [ON DELETE action] ]

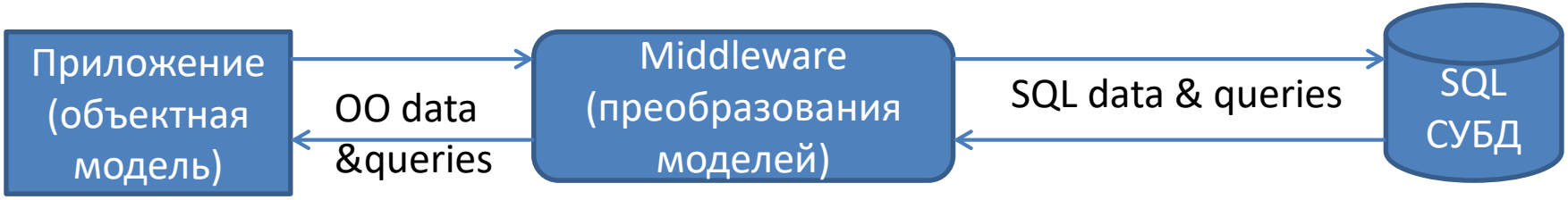
С типизированной таблицей можно обращаться, как с традиционной таблицей, считая, что у нее имеются неявно определенные столбцы (атрибуты структурного типа), а можно относиться к строкам типизированной таблицы, как к объектам соответствующего структурного типа, OID которых содержатся в «самоссылающемся» столбце.

# Пример использования типизированных таблиц

```
CREATE TYPE EMP_T AS (
        EMP_NAME VARCHAR( 200 ),
        EMP_BDATE DATE,
        EMP_SALARY SALARY,
        DEPT_ID REF( DEPT_T ),
        PRO_ID REF( PRO_T )
) INSTANTIABLE NOT FINAL
REF IS SYSTEM GENERATED;

CREATE TYPE PROG_T UNDER EMP_T AS (
        PROG_LANG_SKILL VARCHAR( 100 )
) INSTANTIABLE NOT FINAL;

CREATE TABLE EMPLOYEE OF EMP_T( REF IS EMP_ID SYSTEM GENERATED );

CREATE TABLE PROGRAMMER OF PROG_T UNDER EMPLOYEE;
```

# Основные подходы к объектно-реляционному отображению

Реализации: Oracle, IBM DB2, Informix, PostgreSQL (частично).
Очень многие разработчики приложений используют промежуточное программное обеспечение (middleware) объектно-реляционного отображения, которое само взаимодействует с базами данных на основе БАЗОВОГО подмножества языка SQL без использования каких-либо объектных расширений.




Основные подходы к построению middleware:

1. Использование OO API (JDBC, ADO.NET), которые обеспечивают доступ к реляционным данным и их извлечение в форме, более привлекательной для разработчиков объектно-ориентированных приложений
2. Объекты подстраиваются под реляционную модель (известные методики преобразования, нормализация, BLOB-стратегия)
3. В БД сохраняются не только состояния объектов, но и метаданные, описывающие их структуру (middleware реализует концепцию объектного кэша)

# Истинная реляционная модель данных

Ключевая идея третьего манифеста – чтобы достичь требуемой объектной функциональности, не надо абсолютно ничего делать с реляционной моделью; на основе идей Э. Кодда можно реализовать СУБД, обеспечивающие возможности по части представления и хранения данных произвольно сложной структуры, не меньшие тех, которые обеспечивают объектные и SQL-ориентированные СУБД.
Основное препятствие – тезис Кодда о нормализации отношений (1НФ): в реляционной базе данных должны содержаться только отношения, атрибуты которых определены на доменах, элементы которых являются атомарными (не составными) значениями.

К. Дейт: «Я согласен с Коддом, что желательно оставаться в рамках логики первого порядка, если это возможно. В то же время я отвергаю идею "атомарных значений", по крайней мере, в смысле абсолютной атомарности. В Третьем манифесте мы допускаем наличие доменов, содержащих значения произвольной сложности. Они могут быть даже отношениями. Тем не менее, мы остаемся в рамках логики первого порядка.»

# Типы данных истинной реляционной модели

Три категории типов данных: скалярные типы, кортежные типы и типы отношений.

Скалярный тип – инкапсулированный тип, реальная внутренняя структура которого скрыта от пользователей. Предлагаются механизмы определения новых скалярных типов и операций над ними. Типом атрибута определяемого скалярного типа может являться любой определенный к этому моменту скалярный тип, кортежный тип или тип отношения.

Некоторые базовые скалярные типы данных должны быть предопределены в системе. В число этих типов должен входить тип truth value (булевский тип) с двумя значениями true и false.

Кортежный тип – тип данных, определяемый с помощью генератора типа TUPLE с указанием множества пар <имя_атрибута, тип_атрибута> (заголовка кортежа). Типом атрибута кортежного типа может являться любой определенный к этому моменту скалярный тип, любой кортежный тип и тип отношения. Значением кортежного типа является кортеж, представляющий собой множество триплетов <имя_атрибута, тип_атрибута, значение_атрибута>, которое соответствует заголовку кортежа этого типа.

# Типы данных истинной реляционной модели

Тип отношения – тип данных, определяемый с помощью генератора типа RELATION с указанием некоторого заголовка кортежа. Значением типа отношения является заголовок отношения, совпадающий с заголовком кортежа этого типа отношения, и тело отношения, представляющее собой множество кортежей, соответствующих этому заголовку.
Кортежные типы и типы отношений не являются инкапсулированными: имеется возможность прямого доступа к атрибутам. Для всех разновидностей типов данных поддерживается модель множественного наследования, позволяющая определять новые типы данных на основе уже определенных типов.
База данных – набор долговременно хранимых именованных переменных отношений, каждая из которых определена на некотором типе отношений.

Третий манифест: «Каждый кортеж в отношении *R* содержит в точности одно значение *v* для каждого атрибута *A* в заголовке отношения *H*. Иными словами, *R* находится в первой нормальной форме, 1NF».

# Манипулирование данными и поддержка целостности данных

Эталонные средства манипулирования данными: реляционная алгебра Кодда, реляционная алгебра A.
Языковые средства – язык запросов D:
• для выражения запросов используется алгебраический подход
• запросы, адресуемые к сложным данным, формулируются более точно, чем на SQL
• это же касается сложных операций обновления
• язык обладает вычислительной полнотой
• язык претендует на то, чтобы стать открытым стандартом и заменить SQL.

Любое условное выражение, которое является замкнутой правильно построенной формулой (WFF) реляционного исчисления, должно быть допустимо в качестве спецификации ограничения целостности.
Обязательное требование – определение хотя бы одного возможного ключа для каждой переменной отношения.
Средства поддержки декларативной ссылочной целостности (ограничения внешнего ключа) фигурируют только в разделе рекомендуемых возможностей.

# Реализации истинной реляционной модели

Единственное коммерческое решение:
Dataphor (разработка: 1999-2001, с 2001 по 2008: коммерческий продукт компании Alphora, после ее приобретения Database Consulting Group продукт выпускается под открытой лицензией, текущий релиз в 2018 году, исходный код на C#)

Открытые проекты (университеты и индивидуальные разработчики):

- Alf (Ruby)
- Dee (Python)
- DuroDBMS (multiple languages)
- Rel (Java)
- TclRAL (TCL)