Базы данных

Лекция 04 – Реляционная модель данных

Преподаватель: Поденок Леонид Петрович, 505a-5

+375 17 293 8039 (505a-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2024

2024.02.27

Оглавление

Дореляционные модели данных

Иерархическая модель данных

Иерархическая модель данных возникла из практики работы с данными и создавалась по сути дела программистами. Стандартов на иерархические модели не существует.

Наиболее распространенной СУБД реализующей иерархическую модель являлась СУБД IMS.

Ключевым фактором при создании IMS была необходимость управления большим количеством деталей, связанных друг с другом иерархическим образом (детали → узлы → модули и т.п.).

Иерархическая модель – модель данных на основе записей, в которой все отношения между объектами структурированы в виде деревьев:

1) все записи хранятся в общей древовидной структуре, имеющей одну корневую запись;
2) каждый узел дерева – запись, описывающая экземпляр объекта;
3) соединения между записями в дереве определяют связи между экземплярами объектов.

В иерархической модели связи называются отношениями «предок-потомок», а записи – сегментами.

Потомок может быть связан только с одним предком и для реализации связи с разными предками требуется построение разных деревьев.

Отношение «предок-потомок» реализуют связи «один-ко-многим»;

Иерархическая БД – совокупность деревьев.

Все отношения, которые могут быть представлены в иерархической модели данных – бинарные.

Бинарное отношение — отношение между двумя множествами A и B, то есть всякое подмножество декартова произведения этих множеств: $R \subseteq A \times B$.

Бинарное отношение на множестве A – любое подмножество $R \subseteq A \times A$ (равенство, неравенство, эквивалентность, отношение порядка, ...).

Алгоритм перевода ER-диаграммы в иерархическую модель

Алгоритм перевода ER-диаграммы в иерархическую модель данных представлен следующими шагами:

1) Для каждого объекта ER-диаграммы создается тип сегмента в иерархической модели.

Название типа сегмента соответствует названию объекта.

Атрибуты объекта задают поля записи сегмента (название, тип данных).

2) В диаграммах деревьев отношения **один-ко-многим** задаются с помощью отношений «предок-потомок», причем объект со стороны много в связи становится потомком, а объект со стороны один – предком.

3) Для реализации отношений **многие-ко-многим** создается два разных отношения «предок-потомок», в этих отношениях один и тот же объект выступает в качестве предка (первое отношение «предок-потомок») и в качестве потомка (второе отношение «предок-потомок»).

В одном из таких отношений «предок-потомок» реальные данные для экономии места заменяют на ссылки (указатели) на данные из второго отношения.

4) Если отношение мощности многие-ко-многим на ER-диаграмме обладает атрибутом, то в диаграммах деревьев создается сегмент пересечения, который располагается между предком и потомком и содержит значение атрибута.

Основное достоинство иерархической модели данных — представление данных в виде иерархических связей и автоматическое ограничения целостности (не существует потомков без родителей).

К **основным недостаткам** иерархической модели данных относятся:

- нет разделения логической и физической модели данных;
- непредвиденные запросы требуют реорганизации БД;
- для не иерархических связей описание сильно усложняется и требует много памяти;
- сложность составления программ обработки данных;
- нет стандарта на построение подобных систем.

Сетевая модель данных

Сетевая модель данных – модель данных на основе записей, в которой данные представлены сетевыми структурами типа направленного графа.

Разница между *иерархической моделью данных* и *сетевой* состоит в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков.

Узлы в такой модели представляют объекты и называются – *наборами типов записей данных*.

Ребра графа представляют отношения между объектами и называются *наборами типов связей*.

Наборы типов связей выражают отношения «один-ко-многим» или «один-к-одному» между двумя типами записей, при этом выделяют тип записи *владельца* – со стороны «один» связи, и тип записи *члена* набора – со стороны «много» связи. Сетевая БД состоит из *наборов отношений*.

Все отношения, которые могут быть представлены в сетевой модели данных – бинарные.

Алгоритм перевода ER-диаграммы в сетевую модель данных:

1) Для каждого объекта ER-диаграммы создается тип записей в сетевой модели.

Название типа записей соответствует названию объекта.

Атрибуты объекта задают поля типа записи (название, тип данных).

2) Для отношения «**один-ко-многим**» тип записи со стороны «один» отношения становится владельцем набора, а тип записи со стороны «много» отношения – членом набора.

Если мощность отношения «**один-к-одному**», то типы записей в наборе выбираются произвольно.

3) Для каждого n-арного отношения, где n > 2, создается связывающая запись, которая становится типом записи члена для n наборов.

Владельцами наборов назначаются типы записей со стороны «один», полученных в результате преобразований отношений «один-ко-многим».

4) Для реализации отношений «**многие-ко-многим**» создается связывающая запись, которая становится типом записи члена для двух наборов, владельцами которых являются типы записей, соответствующие объектам связи.

5) Если отношение мощности «многие-ко-многим» на ER-диаграмме обладает атрибутом, то этот атрибут добавляется в связывающую запись.

Основные достоинства сетевой модели данных:

- высокая производительность в статичных системах с простой структурой;
- наличие ограничений целостности в отношении «зависших» потомков;
- **наличие стандарта (DBTG).**

К **основным недостаткам** сетевой модели данных относятся:

- структура БД определяется запросами пользователей – смена запросов ведет к реорганизации всей БД;
- производительность непредвиденных приложений низкая – нет возможности расширения;
- определение схемы влияет на физический уровень БД (формат записей);
- сложность составления программ обработки данных;
- сложно установить ограничения на образование связей.

Операции

- вставка, чтение, обновление и удаление записей;
- вставка, модификация (переключение) и удаление связей между владельцем и членом.

Реляционная модель данных

Замечание о терминологии

SQL и реляционная модель – вовсе не одно и то же.

Дейт уточняет:

> **«Если знания о реляционной модели проистекают исключительно из знания SQL, то эти знания могут оказаться неверными. Отсюда, в частности, следует, что кое-что из уже известного следует забыть.»**

В рамках реляционной теории употребляются формальные термины — «*отношение*», «*кортеж*» и «*атрибут*».

В SQL эти термины не используются, там применяются более «дружественные» слова — *таблицы*, *строка* и *столбец/колонка*.

Истина заключается в том, что отношение – это не таблица, кортеж – не строка, а атрибут – не столбец.

SQL пытался упростить один набор терминов, но он сделал все возможное для усложнения другого. Имеются в виду термины *оператор*, *функция*, *процедура*, *подпрограмма* и *метод* — все они обозначают по существу одно и то же (быть может, с незначительными вариациями).

Реляционная модель для всего вышеперечисленного использует термин **оператор**.

Представление данных в виде деревьев, сетевых и списковых структур в общем случае препятствует многим изменениям данных, необходимым при росте базы.

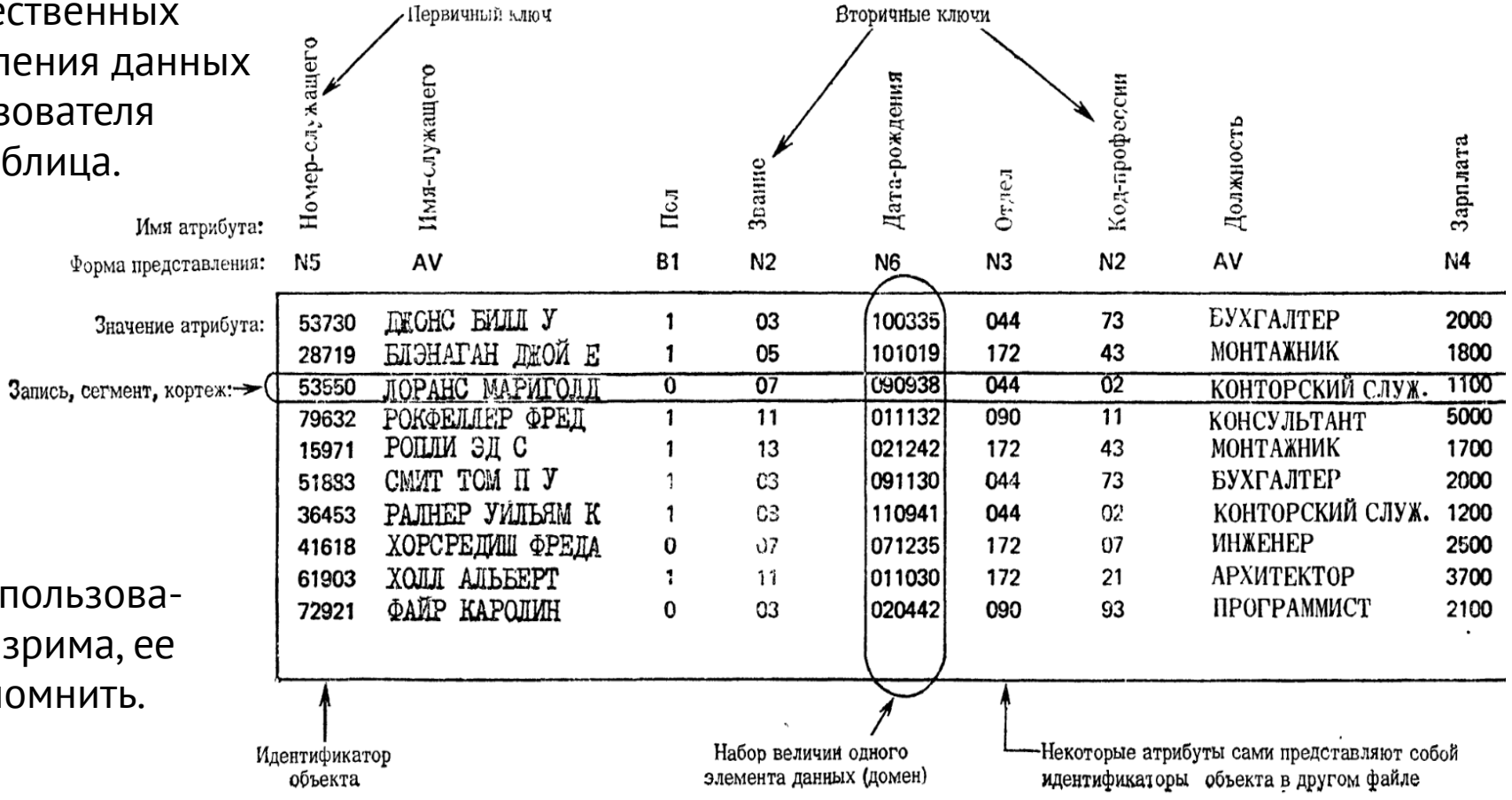
Рост базы может привести к нарушению логического представления данных и, как следствие, к изменениям в прикладных программах.

Избежать растущей сложности древовидных и сетевых структур можно с помощью метода, называемого *нормализацией*. Этот метод был разработан и активно применялся Коддом.

Нормализация относится к представлению данных пользователем или к логическому описанию данных; она не связана с физическим представлением данных.

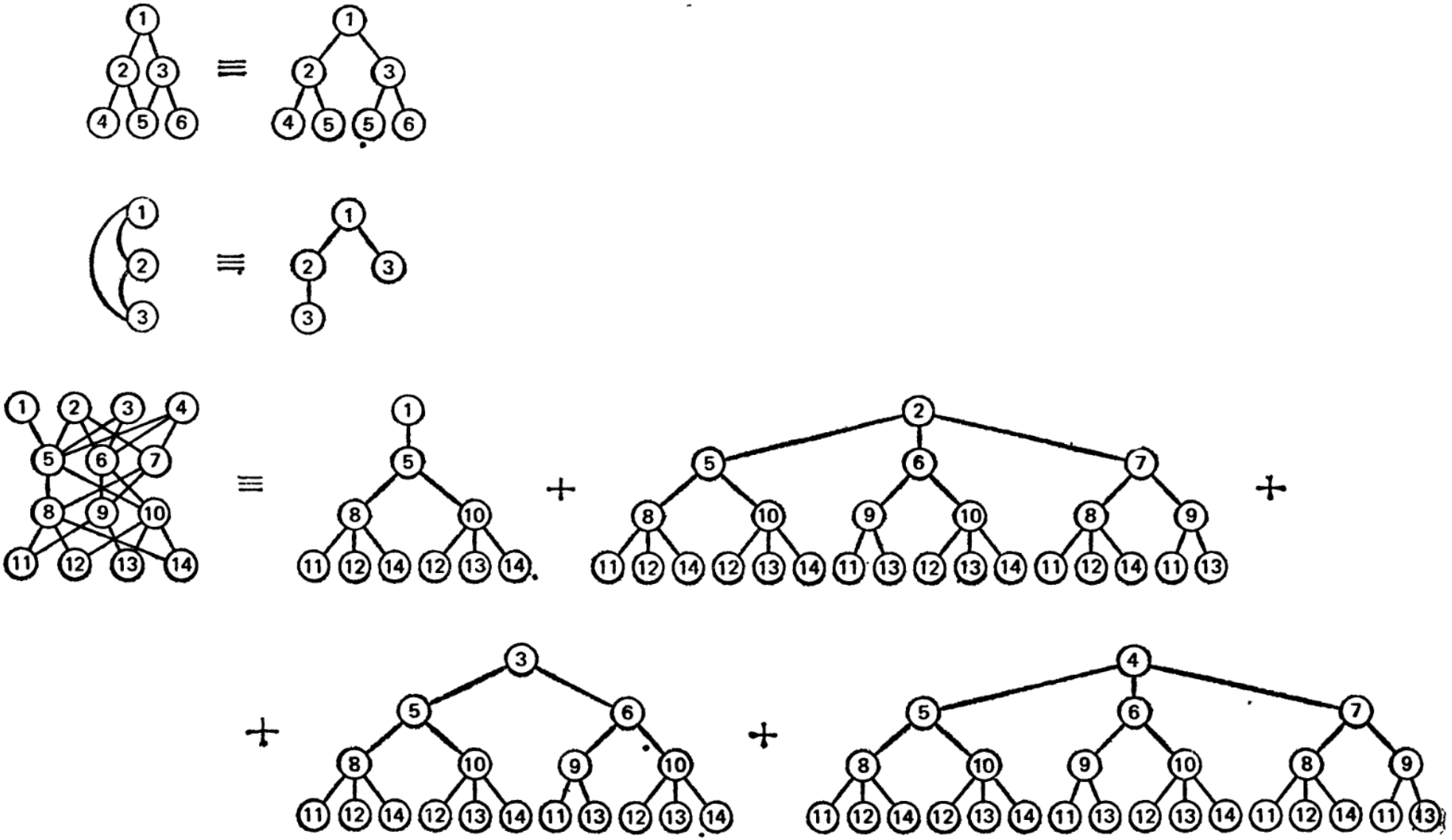
Табличное представление

Один из самых естественных способов представления данных для обычного пользователя — это двумерная таблица.



Она привычна для пользователя, понятна и обозрима, ее структуру легко запомнить.

Любая сетевая структура может быть с *некоторой избыточностью* разложена в совокупность древовидных структур.



Любое представление данных может быть сведено с *некоторой избыточностью* к двумерным плоским файлам.

Связи между данными могут быть представлены в форме двумерных таблиц.

Этот процесс, выполняемый шаг за шагом для каждой связи между данными в базе, называется *нормализацией*.

Таблицы могут быть построены таким образом, что не будет утеряна информация о связях между элементами данных.

Такие таблицы — это прямоугольные массивы, которые можно описать математически.

Таблица обладает следующими свойствами

1) Каждый элемент таблицы представляет собой один элемент данных – повторяющиеся группы отсутствуют.

2) Все столбцы в таблице однородные – элементы одного столбца имеют одинаковую природу.

3) Столбцам однозначно присвоены имена.

4) В таблице нет двух одинаковых строк.

5) В операциях с такой таблицей ее строки и столбцы могут просматриваться в любом порядке и в любой последовательности безотносительно к их информационному содержанию и смыслу.

Выделение повторяющейся группы в отдельную таблицу путем расщепления

Заказ-на-закупку

No заказа	No поставщика	Дата заказа	Дата поставки	No изделия	Цена	Кол-во	ИТОГО

Заказ-на-закупку

No заказа	No поставщика	Дата заказа	Дата поставки	ИТОГО

Ключ

Партия-товара

No заказа	**No изделия**	Цена	Кол-во

Ключ

Ключи

Каждый кортеж должен иметь ключ — уникальный идентификатор.

Иногда кортеж может идентифицироваться значением одного атрибута. Например, атрибут НОМЕР-ЗАКАЗА однозначно определяет кортеж файла ЗАКАЗ-НА-ЗАКУПКУ.

Но бывает и так, что для идентификации кортежа необходимы значения нескольких атрибутов.

Для определения кортежа файла ПАРТИЯ-ТОВАРА недостаточно одного атрибута, и ключ должен состоять из двух атрибутов:

НОМЕР-ЗАКАЗА + НОМЕР-ИЗДЕЛИЯ.

Ключ должен обладать двумя свойствами:

1) Однозначная идентификация кортежа — кортеж должен однозначно определяться значением ключа.

2) Отсутствие избыточности — никакой атрибут нельзя удалить из ключа, не нарушая при этом свойства однозначной идентификации.

Для кортежей одного отношения может существовать несколько наборов атрибутов, удовлетворяющих двум приведенным свойствам.

Такие наборы называются ***возможными ключами***.

Тот ключ, который фактически будет использоваться для идентификации записей, называется ***первичным ключом***.

Правила выбора атрибутов первичного ключа

Для первичного ключа атрибуты следует выбирать так, чтобы для них был заранее известен диапазон значений, а их количество было бы как можно меньше.

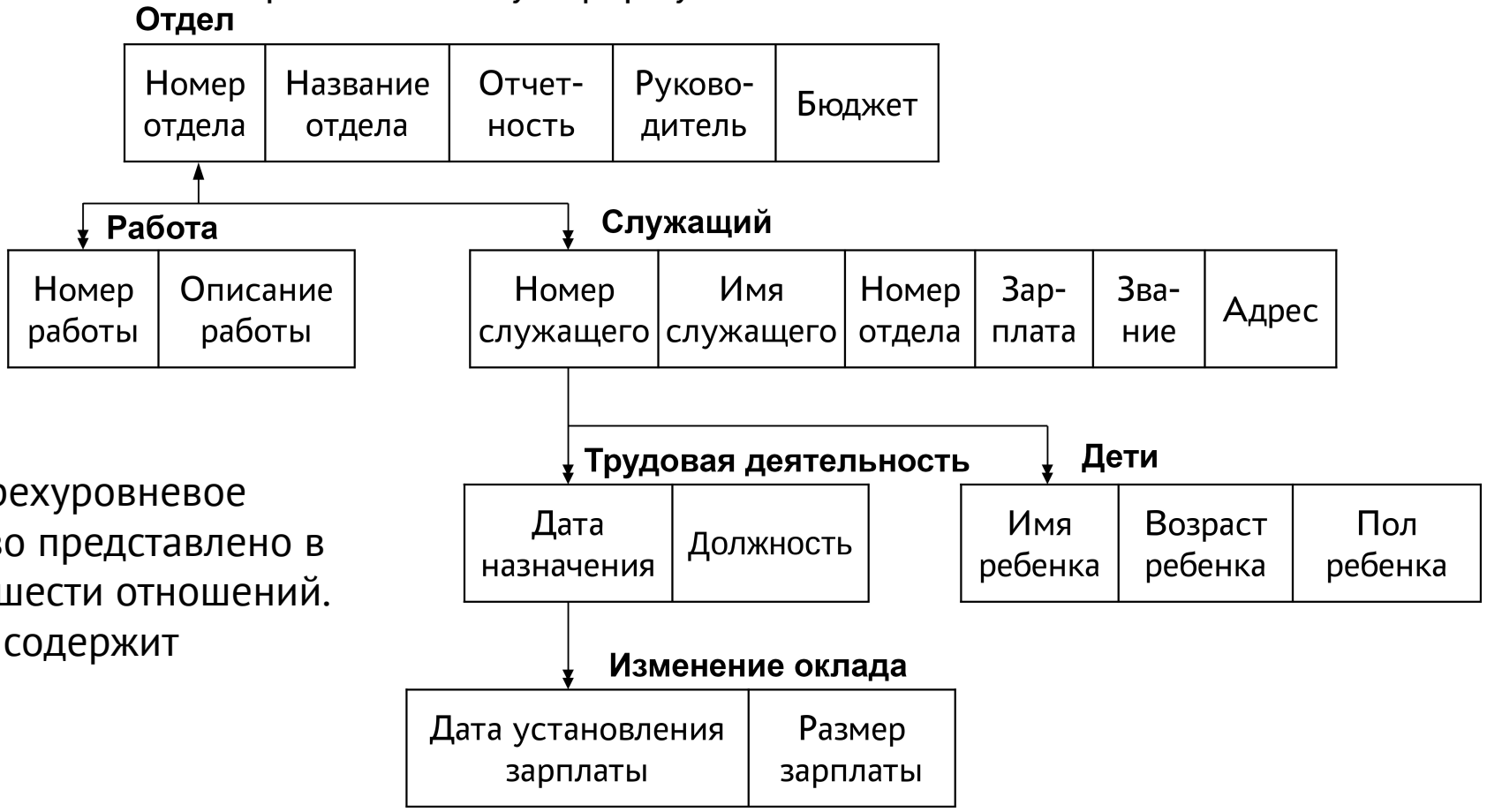
Обозначения

Заказ-на-закупку(Номер-заказа, Номер-поставщика, Дата-заказа, Дата-поставки, Итого)

Партия-товара(Номер-заказа, Номер-изделия, Цена, Количество)

Древовидные структуры

Преобразование в нормализованную форму с помощью добавления ключевых элементов



Четырехуровневое дерево представлено в виде шести отношений. Ключ содержит

Отдел(Номер-отдела, Название-отдела, Отчетность, Руководитель, Бюджет)
Работа(**Номер-отдела**, Номер-работы, Описание-работы)
Служащий(Номер-служащего, Имя-служащего, Номер-отдела, Зарплата, Звание, Адрес)
Изменение-оклада(**Номер-служащего**, Дата-установления-зарплаты, Размер-зарплаты)
Дети(**Номер-служащего**, Имя-ребенка, Возраст-ребенка, Пол-ребенка)
Трудовая-деятельность(**Номер-служащего**, Дата-назначения, Должность)

Реляционные базы данных

Сторонники нормализации ввели собственную терминологию и называют простые вещи специальными словами.

Таблица такого вида, как представлено выше, называется *отношением*.

База данных, построенная с помощью отношений, называется *реляционной базой данных*.

Таким образом, реляционная база данных строится из «плоских» (двумерных) наборов элементов данных.

Отношение, или таблица,— это набор *кортежей*.

Кортежи являются **n**-мерными, т. е. если таблица имеет **n** столбцов,

Отношение в этом случае называется *отношением степени* **n**.

Отношение степени **2** называется *бинарным*, отношение степени **3** — *тернарным*, а отношение степени **n** — *n-арным*.

Набор значений элементов данных одного типа, т. е. один столбец таблицы, называется *доменом*.

Столбец с номером **k** называется **k-м** *доменом отношения*.

В математике R определяется как отношение, заданное на n множествах S_1, S_2, …, S_n (не обязательно различных), если оно представляет собой набор кортежей, таких, что первый элемент каждого кортежа есть элемент из S_1, второй — элемент из S_2 и т. д.

Для описания таких отношений и операций над ними существуют точные математические обозначения, основанные на алгебре отношений или исчислении отношений (реляционная алгебра).

Реляционная алгебра — замкнутая система операций над отношениями в реляционной модели данных.

Операции реляционной алгебры также называют *реляционными операциями*.

Кортеж — определение

Пусть дана коллекция типов $T_i(i = 1, 2, ..., n)$, которые не обязательно все должны быть разными.

Значением кортежа (или кратко кортежем), определенным с помощью этих типов (назовем его t), является множество упорядоченных троек в форме $< A_i, T_i, v_i >$, где A_i — имя атрибута, T_i — имя типа и v_i —значение типа T_i.

Кроме того, кортеж t должен соответствовать приведенным ниже требованиям:

- Значение n является ***степенью***, или ***арностью*** кортежа t.
- Упорядоченная тройка $< A_i, T_i, v_i >$ является компонентом (частью) кортежа t.
- Упорядоченная пара $< A_i, T_i >$ представляет собой ***атрибут*** t и однозначно определяется именем атрибута A_i (имена атрибутов A_i и A_j совпадают, только если $i = j$).
- Значение v_i — это ***значение атрибута***, соответствующее атрибуту A_i кортежа t.
- Тип T_i — это соответствующий ***тип атрибута***.
- Полное множество атрибутов составляет ***заголовок кортежа*** t.
- Тип кортежа t определен заголовком t, а сам заголовок и этот тип кортежа имеют такие же атрибуты (и поэтому такие же имена и типы атрибутов) и такую же степень, как и кортеж t.

Точное определение имени типа кортежа:

```
TUPLE {A1 T1, A2 T2, ..., An Tn}
```

Пример кортежа:

MAJOR_PNO : PARTNO	MINOR_PNO : PARTNO	QTY : QUANTITY
P2	P4	7

Имена атрибутов: MAJOR_PNO, MINOR_PNO, QTY
Имена типов: PARTNO, QUANTITY
Значения: PARTNO{'P2'}, PARTNO{'P4'}, QUANTITY{7}

TUPLE {MAJOR_PNO PARTNO, MINOR_PNO PARTNO, QTY QUANTITY}

В неформальном изложении принято исключать имена типов из заголовка кортежа и показывать только имена атрибутов.

Поэтому показанный выше кортеж может быть неформально представлен, как:

MAJOR_PNO	MINOR_PNO	QTY
P2	P4	7

Термин «кортеж» *приблизительно соответствует* понятию строки, так же, как термин «отношение» *приблизительно соответствует* понятию таблицы.

В реляционной модели не используется термин «поле», вместо него используется термин «атрибут», который в рассматриваемом контексте *примерно соответствует* понятию столбца таблицы.

Пусть есть БД отделов и сотрудников:

DEPT (Отделы)

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP (Служащие)

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finci	D2	30K
E4	Saito	D2	35K

Таблицы **DEPT** И **EMP** в базе данных фактически являются ***переменными отношения***.

Соответственно, как любые переменные, они имеют значения. Их значения — ***значения отношения***, которые они принимают в разное время.

Предположим, например, что таблица **EMP** в данный момент имеет значение (значение отношения), которое показано выше, и допустим, что мы удалили строку о сотруднике с фамилией Saito.

```
DELETE EMP WHERE EMP# = EMP# ('E4') ;
```

Результат выполнения этой операции:

EMP (Служащие)

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finci	D2	30K

Концептуально это действие можно описать таким образом — **старое значение отношения EMP было заменено в целом совершенно другим, новым значением отношения.**

Старое значение с четырьмя строками и новое значение с тремя очень похожи, но в действительности они являются разными.

Можно считать, что данная операция удаления строки — это просто альтернативный, упрощенный способ записи для определенной операции реляционного присваивания, которая могла бы выглядеть примерно следующим образом[1].

```
EMP := EMP WHERE NOT (EMP# = EMP#('E4'));
```

Сначала вычисляется выражение, расположенное справа от знака присваивания, а затем его значение присваивается переменной, которая записана перед знаком присваивания.

В целом задача заключается в том, чтобы заменить «старое» значение отношения EMP «новым».

Таким же образом операции INSERT и UPDATE также являются просто сокращенной формой записи соответствующих реляционных операций присваивания.

В реляционной теории (РТ) следует четко различать *переменные отношения* и сами *отношения*.

Для переменной отношения (*relation variable*) иногда используется термин *relvar*[2].

Термин «*переменная отношения*» (relvar) не является общепринятым и в документации на БД практически не встречается, хотя различать сами отношения, т.е. значения отношений, и переменные отношения как таковые, с точки зрения РТ очень важно. Например, ограничения целостности и операции обновления, применяются к переменным отношения, а не к отношениям.

---

1) Оператор DELETE и равносильный ему оператор присваивания записаны на языке Tutorial D

2) Различие между значениями отношения и переменными отношения фактически представляют собой особый случай различия между значениями и переменными в целом.

Смысл отношений

Столбцы в отношениях связаны с типами данных. Реляционная модель допускает неограниченный набор типов [данных]. Это означает, что пользователи могут как применять определяемые системой или встроенные типы, так и определять собственные. Например, определять типы можно так («...» здесь заменяет сами определения, которые на сей момент не важны).

```
TYPE EMP_NO ... ;
TYPE NAME   ... ;
TYPE DPT_NO ... ;
TYPE MONEY  ... ;
```

Тип **EMP_NO**, например, можно рассматривать, как множество всевозможных табельных номеров, тип **NAME** — как множество всевозможных имен и т.д.

Каждое отношение (каждое значение отношения) состоит из двух частей:

1) набор пар (<имя_столбца> : <имя_типа>) — заголовок;

2) набор строк, согласованных с этим заголовком — тело.

Пример — часть отношения EMP, дополненного обозначениями типов столбцов.

EMP (Служащие)

EMP# : EMP_NO	ENAME : NAME	DEPT# : DEPT_NO	SALARY : MONEY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finci	D2	30K
E4	Saito	D2	35K

Хотя на практике компоненты заголовка <имя_типа> обычно опускают, необходимо учитывать, что концептуально они всегда присутствуют.

Отношения можно представлять также следующим образом:

1) В определенном отношении **R** его заголовок представляет собой некоторый ***предикат*** (функция, возвращающая значения истинности и принимающая ряд формальных параметров).

2) Каждая строка в теле отношения R представляет собой определенное ***истинное высказывание***, полученное из предиката путем подстановки определенных значений фактических параметров соответствующего типа вместо формальных параметров этого предиката (путем конкретизации).

EMP (Служащие)

EMP# : EMP_NO	ENAME : NAME	DEPT# : DEPT_NO	SALARY : MONEY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
...	...	...	...

Для отношения EMP предикат будет следующим:

«Служащий с табельным номером **EMP#** и фамилией **ENAME** работает в отделе с номером **DEPT#** и получает зарплату **SALARY**».

Здесь формальные параметры – EMP#, ENAME, DEPT# И SALARY. Они соответствуют четырем столбцам переменной отношения **EMP**.

Соответствующие истинные утверждения:

«Служащий с табельным номером E1 и фамилией Lopez работает в отделе с номером D1 и получает зарплату 40 тыс. долл. в год.»

«Служащий с номером E2 и фамилией Cheng работает в отделе с номером D1 и получает зарплату 42 тыс. долл. в год.»

Иными словами, **типы** — это объекты (множества объектов), которые могут стать предметом обсуждения; **отношения** — это факты (множества фактов), касающиеся объектов, которые могут стать предметом обсуждения.

Типы связаны с отношениями точно так же, как существительные связаны с предложениями на естественном языке.

В примере, приведенном выше, предметом обсуждения являются:

- табельные номера служащих EMP_NO;
- имена NAME;
- номера отделов DEPT_NO;
- значения денежных сумм MONEY.

Об обсуждаемом же предмете можно привести истинное высказывание следующего вида:

«Служащий с определенным табельным номером имеет определенное имя, работает в определенном отделе и получает определенную зарплату».

Структурные свойства

В оригинальной модели три основных компонента:
- структура;
- целостность;
- средства манипулирования.

Структура

Главным структурным свойством является само отношение.
Отношения принято изображать на бумаге в виде таблиц.

DEPT

DNO	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

ENO	ENAME	DNO	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finci	D2	30K
E4	Saito	D3	35K

Δ

DEPT.DNO ссылается на EMP.DNO

Отношения определены над *типами* (типы называют также *доменами*).

Тип представляет собой *концептуальное множество значений*, которые могут принимать фактические (конкретные) значения атрибутов фактических (конкретных) отношений.

В простой БД об отделах и служащих можно выделить тип **DNO** («номера отделов»), представляющий все допустимые номера отделов, и тогда атрибут **DNO** в отношении **DEPT** и атрибут **DNO** в отношении **EMP** будут принимать значения из соответствующего ему концептуального множества.

Атрибуты не обязательно называть так же, как соответствующий тип.

Структурный аспект

Данные в базе воспринимаются пользователем, как таблицы и никак иначе.

Аспект целостности

Эти таблицы отвечают определенным условиям целостности.

Аспект обработки

В распоряжении пользователя имеются операторы манипулирования таблицами, например, предназначенные для поиска данных, которые генерируют новые таблицы на основании уже имеющихся и среди которых есть, по крайней мере, операторы:

- сокращения (restrict);
- проекции (project);
- объединения (join).

Операция сокращения извлекает указанные строки из таблицы. В названии этой операции подразумевается, что *кардинальность* ее результата меньше или равна кардинальности исходной таблицы.

Операцию сокращения иногда называют *выборкой* (select). Однако, соответствующий оператор соответствует оператору SELECT языка SQL не полностью.

Операция проекции предназначена для извлечения определенных столбцов из таблицы.

Операция соединения предназначена для получения комбинации двух таблиц на основе общих значений в общих столбцах.

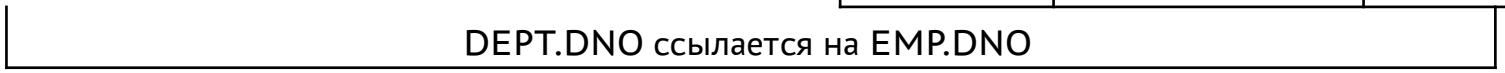
Операция соединения

DEPT

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M
D3	Research	5M

EMP

EMP#	ENAME	DEPT#	SALARY
E1	Lopez	D1	40K
E2	Cheng	D1	42K
E3	Finci	D2	30K
E4	Saito	D2	35K



В таблицах DEPT и EMP есть общий столбец DEPT#, а следовательно, для этих таблиц можно выполнить операцию соединения на основе общих значений в этом столбце. При выполнении данной операции строка таблицы DEPT соединяется со строкой таблицы EMP и образуется более длинная строка, но подобное происходит тогда и только тогда, когда эти две строки имеют одинаковое значение поля DEPT#.

Например, можно соединить в результирующую строку следующие строки таблиц DEPT и EMP (названия столбцов приведены для наглядности). Это возможно, так как в общем столбце рассматриваемых строк имеется одно и то же значение D1.

DEPT#	DNAME	BUDGET	**EMP#**	ENAME	SALARY
D1	Marketing	10M	E1	Lopez	40K
D1	Marketing	10M	E2	Cheng	42K
D2	Development	12M	E3	Finci	30K
D2	Development	12M	E4	Saito	35K

Общий результат состоит из множества всех таких соединенных строк.

Столбец DEPT# в каждой результирующей строке встречается один раз, а не два.

Следует также отметить, что в поле DEPT# таблицы EMP отсутствует значение D3, т.е. нет служащих, работающих в отделе D3, поэтому в данном поле нет и результирующих строк со значением D3, хотя строка со значением D3 в таблице DEPT присутствует.

Сокращение (извлекает указанные строки из таблицы)

DEPTs where BUDGET > 8M

DEPT#	DNAME	BUDGET
D1	Marketing	10M
D2	Development	12M

Проекция (извлекает определенные столбцы из таблицы)

DEPTs over DEPT#, BUDGET

DEPT#	BUDGET
D1	10M
D2	12M
D3	5M

Важная особенность — результат выполнения каждой из трех представленных операций — это еще одна таблица, т.е. операторы сокращения, проекции и соединения порождают таблицы из таблиц.

Это — пример проявления алгебраического свойства замкнутости. Оно имеет очень большое значение, главным образом потому, что результатом выполнения операции является объект того же рода, что и объект, над которым производилась операция, а именно — таблица.

Это значит, что

К результатам операций можно снова применить какую-либо операцию.

Например, можно сформировать проекцию соединения, соединение двух сокращений, сокращение проекции и т.д.

То есть, можно использовать вложенные выражения, т.е. выражения, в которых операнды представлены выражениями, а не простыми именами таблиц.

Промежуточные результаты

Тот факт, что результатом выполнения каждой операции является таблица, не подразумевает, что система обязательно должна полностью материализовать результат каждой отдельной операции в виде реальной таблицы — промежуточный результат, который создается операцией соединения, возможно, никогда и не будет существовать в виде полной материализованной таблицы, как таковой.

На практике, как правило, разработчики каждой системы настойчиво стремятся избежать полной материализации промежуточных результатов в целях повышения производительности.

Если промежуточные результаты материализуются полностью, то стратегия вычисления выражения в целом называется ***материализованными вычислениями*** (materialized evaluation).

Если промежуточные результаты передаются последующей операции по частям, то этот подход называется ***конвейерными вычислениями*** (pipelined evaluation).

Важно — операции применяются сразу ко всему множеству строк, а не к отдельной строке за один раз, т.е. операндами и результатами являются не отдельные строки, а целые таблицы, которые содержат множество строк.

Особенности

Определение реляционной системы требует, чтобы база данных только воспринималась пользователем как набор таблиц.

Таблицы в реляционной системе являются ***логическими***, а не ***физическими*** структурами.

На самом деле, на физическом уровне система может использовать любую из существующих структур памяти (последовательный файл, индексирование, хэширование, цепочку указателей, сжатие и т.п.), лишь бы существовала возможность отображать эти структуры в виде таблиц на логическом уровне.

Таблицы представляют собой ***абстракцию*** способа физического хранения данных, в которой все нюансы реализации на уровне физической памяти скрыты от пользователя, в частности это:

1) размещение хранимых записей;
2) упорядочение хранимых записей;
3) кодировка хранимых данных;
4) префиксы хранимых записей;
5) хранимые структуры доступа, такие как индексы и т.д.

Термин «логическая структура» в терминологии ANSI/SPARC (три уровня) относится как к концептуальному, так и ко внешнему уровням:

- концептуальный, и внешний уровни в реляционной системе являются одинаково реляционными;
- внутренний или физический уровень реляционными не является.

Реляционная теория не может определить внутренний уровень, она определяет лишь то, как база данных представлена пользователю. Единственное требование состоит в следующем — какая бы физическая структура не была выбрана, она должна полностью реализовывать существующую логическую структуру.

У реляционных баз данных есть одно замечательное свойство, так называемый информационный принцип — все информационное наполнение базы данных представлено одним и только одним способом, а именно — **явным заданием значений, помещенных в позиции столбцов в строках таблицы**. Этот метод представления — единственно возможный для реляционных баз данных (на логическом уровне). В частности, **нет никаких указателей, связывающих одну таблицу с другой**.

Аспект целостности

На практике для БД отделов и сотрудников потребовалось бы определить несколько ограничений поддержки целостности базы. Например:

- зарплата служащих не должна выходить за пределы от 25 до 95 тыс. долл. в год;
- бюджет отдела должен находиться в пределах от 1 до 15 млн.$
- и т.д.

Некоторые из таких правил имеют настолько важное практическое значение, что получили специальные названия.

1) Каждая строка в таблице DEPT должна включать уникальное значение столбца DEPT#; аналогично, каждая строка в таблице EMP должна включать уникальное значение столбца EMP#.

Говорят, что DEPT# и EMP# в таблице EMP являются **первичными ключами для своих таблиц**.

2) Каждое значение столбца DEPT# в таблице EMP **должно** быть представлено и в виде значения столбца DEPT# в таблице DEPT в соответствии с тем фактом, что каждый служащий должен быть приписан к существующему отделу. Говорят, что столбец DEPT# в таблице EMP является **внешним ключом**, ссылающимся на первичный ключ таблицы DEPT .

Оптимизация

Все реляционные операции, такие как сокращение, проекция и соединение, выполняются на уровне множеств. Поэтому реляционные языки часто называют ***непроцедурными***, а ***декларативными***, так как пользователь указывает, ***что*** делать, а не ***как*** делать.

Фактически пользователь сообщает лишь, что ему нужно, без указания процедуры получения результата.

Процесс навигации (перемещения) по хранимой базе данных в целях удовлетворения запроса пользователя выполняется системой автоматически, а не пользователем вручную, поэтому реляционные системы иногда называют ***системами автоматической навигации***.

В ***нереляционных*** системах за навигацию по базе данных в основном несет ответственность сам пользователь.

Следует отметить, что ***непроцедурный*** — это хотя и общепринятый, но не совсем точный термин, потому что *процедурный* и *непроцедурный* — понятия относительные.

Обычно можно с уверенностью определить лишь то, является ли язык А более (или менее) процедурным, чем язык Б.

Поэтому точнее будет сказать, что реляционные языки, такие как SQL, характеризуются более высоким уровнем абстракции, чем типичные языки программирования, подобные C++ или COBOL.

В принципе, именно более высокий уровень абстракции способствует повышению продуктивности труда программистов, свойственному для реляционных систем.

Ответственность за организацию выполнения автоматической навигации возложена на очень важный компонент СУБД — ***оптимизатор***. Его работа заключается в том, чтобы выбрать самый эффективный способ выполнения для каждого запроса пользователя.

Каталог базы данных

Каждая СУБД должна поддерживать функции каталога, или словаря.

Каталог обычно размещается там, где хранятся различные схемы (внешние, концептуальные, внутренние) и все, что относится к отображениям («внешний-концептуальный», «концептуальный-внутренний», «внешний-внешний»).

В каталоге содержится подробная информация, касающаяся различных объектов, имеющих значение для самой системы (описательная информация или метаданные):

- переменные отношения;
- индексы;
- ограничения для поддержки целостности;
- ограничения защиты;
- и пр.

Эта информация необходима для обеспечения правильной работы системы.

Например, оптимизатор использует информацию каталога об индексах и других физических структурах хранения данных, а также прочую информацию, необходимую ему для принятия решения о том, как выполнить тот или иной запрос пользователя.

Подсистема защиты использует информацию каталога о пользователях и установленных ограничениях защиты, чтобы разрешить или запретить выполнение поступившего запроса.

Свойством реляционных систем является то, что их каталог также состоит из переменных отношения, таких же как и пользовательские.

Чтобы отличать их от пользовательских их называют *системными* переменными отношения.

В результате пользователь может обращаться к каталогу так же, как к своим данным.

Например, в каталоге любой системы SQL обычно содержатся системные переменные отношения **`TABLES`** и **`COLUMNS`**, назначение которых — описание известных системе таблиц, т.е. переменных отношения, и столбцов этих таблиц.

Каталог базы данных отделов и служащих может быть схематически представлен в **TABLES** и **COLUMNS** в следующем виде:

TABLE

TAB_NAME	COLS	ROWS	.....
DEPT	3	3	.....
EMP	4	4	.....
TABLE	3	123	.....

COLUMNS

TAB_NAME	COL_NAME	COL_TYPE	.....
DEPT	DEPT#	DPT_NO	.....
DEPT	DNAME	NAME	.....
DEPT	BUDGET	MONEY	.....
EMP	EMP#	EMP_NO	.....
EMP	ENAME	NAME	.....
EMP	DEPT#	DPT_NO	.....
EMP	SALARY	MONEY	.....
TABLE	TAB_NAME	TAB_NAME_T	.....
TABLE	COLS	COLS_T	.....
TABLE	ROWS	ROWS_T	.....

Каталог также должен описывать сам себя, т.е. включать записи, описывающие переменные отношения самого каталога.

Какие столбцы содержит переменная отношения **DEPT**[3]:

```
(COLUMNS WHERE TABNAME = 'DEPT'){COLNAME}
```

В каких переменных отношения есть столбец **EMP#**.

```
(COLUMNS WHERE COLNAME = 'EMP#'){TABNAME}
```

3) предполагается, что по каким-то причинам пользователь не имеет этой информации