Базы данных

Лекция 05 – Реляционная модель данных

Преподаватель: Поденок Леонид Петрович, 505а-5

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

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

Кафедра ЭВМ, 2022

2022.10.08

Оглавление

Реляционная модель данных

Отношения и переменные отношения

Реляционная база данных — это база данных, в которой данные представлены в виде таблиц.
Вопрос: почему такая БД называется именно реляционной, а не, например, табличной?
Ответ: термин «relation» (отношение) — это формальное название *определенного* вида таблиц.
relation — связь, отношение, соотношение, зависимость, ... (перевод с англ. в п.ч.в.)
Например, можно сказать, что база данных отделов и служащих, представленная ниже, содержит два отношения.

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

Δ

EMP.DEPT# ссылается на DEPT.DEPT#

Теория реляционной модели была первоначально сформулирована Коддом (IBM) и при этом умышленно применялись лишь определенные, тщательно подобранные термины, не допускающие многозначности. Причина — многие распространенные в те времена термины были очень нечеткими. Им не хватало точности, необходимой для той формальной теории, которую предложил Кодд.

Например, термин «*запись*». В разное время и в различных контекстах он мог означать экземпляр записи или тип записи, логическую запись или физическую запись, хранимую запись или виртуальную запись, ...

В настоящее время в неформальном контексте термины «отношение» и «таблица» принято считать синонимами. На практике в подобном контексте термин таблица используется гораздо чаще, чем термин отношение.

Дейт в своей книге « Введение в системы баз данных» объясняет, почему термин «отношение» поставлен на первое место.

- реляционные системы основаны на реляционной модели. Реляционная модель — это абстрактная теория данных, основанная на таких областях математики, как теория множеств и логика предикатов (исчисление высказываний). Поэтому в формальной реляционной модели термин «запись» не используется вообще — вместо него применяется термин «кортеж» (tuple), точное определение которого дал сам Кодд, когда ввел его впервые.

Кортеж.

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

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.

Снова обратимся к 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

Δ

EMP.DEPT# ссылается на DEPT. DEPT#

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

Соответственно, как любые переменные, 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».

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.

Например, пользователь сделал следующий запрос (язык Tutorial D).

```
(EMP WHERE EMP# = EMP#('E4')){SALARY}                    (1)
```

Выражение **(EMP WHERE ...)** означает операцию сокращения текущего значения переменной отношения **EMP**, касающаяся той строки, в которой значение столбца **EMP#** равно **E4**.

R{SALARY} означает проекцию отношения **R** по столбцу **SALARY**.

В данном случае используется реляционное свойство замкнутости — выражение (1) является вложенным, в котором результат **R** операции сокращения **EMP** используется в качестве входных данных для операции проекции. Результатом (1) становится отношение, состоящее из одного столбца и одной строки, которое содержит данные о заработке служащего **E4**.

В этом примере могут применяться, как минимум, два способа доступа к необходимым данным:

Наивный — последовательный физический просмотр (хранимой версии) переменной отношения **EMP**, пока требуемая запись не будет найдена.

Индексный — если есть индекс для столбца **EMP#** переменной отношения **EMP**[3], то переход с помощью этого индекса к данным служащего с номером **E4** будет прямым.

Оптимизатор определит, какую из двух возможных стратегий следует применить. В общем случае для реализации любого конкретного реляционного запроса оптимизатор будет осуществлять выбор стратегии, исходя из соображений, подобных следующим:

- на какие переменные отношения есть ссылки в запросе;
- насколько велики эти переменные отношения в настоящее время;
- какие существуют индексы;
- насколько избирательны эти индексы;
- как физически группируются данные на диске;
- какие реляционные операции используются; и т.д.

---

3) этот индекс скорее всего существует, поскольку столбец является уникальным, а большинство систем поддерживает индекс для обеспечения уникальности.

Резюме

Пользователь указывает в запросе, какие данные ему требуются, а не как их получить, а стратегия доступа к данным выбирается оптимизатором (автоматическая навигация).

В результате пользователи и пользовательские программы становятся независимыми от применяемых стратегий доступа и от физического представления данных.

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**[4]:

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

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

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

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

Низкоуровневые операции логического уровня в таблицах

```
TABLE LIST:LST (ID, NAME, DATE, ...)
TABLE ITEM:ITM (ID, LST_ID, PART_NO, NAME, RPOVIDER, QUANT, ...)

PRIMKEY LST:BY_ID (ID)                  # LST:ID
PRIMKEY ITM:BY_ID (ID)                  # ITM:ID

INDEX LST:BY_NAME         (NAME)
INDEX LST:BY_DATE         (DATE)
INDEX ITM:BY_NAME         (NAME)
INDEX ITM:BY_NAME_INLIST (LST_ID, NAME)
INDEX ITM:BY_PNO_INLIST  (LST_ID, PART_NO)  # кандидат в PrimKey (?поставщик?)
```

Простейшие сериальные операции

```
SET(TABLE) – установка курсора в «сыром» порядке записей
NEXT(TABLE) – просмотр таблицы в направлении с начала к концу
PREV(TABLE) – просмотр таблицы в направлении с конца к началу
INSERT(RECORD, TABLE) – вставка в таблицу без обновления индексов
```

Прямой доступ по первичному ключу

GET(RECORD, TABLE, KEY) — получить строку (запись) из таблицы по значению перв. ключа
PUT(RECORD, TABLE, KEY) — обновить строку (запись) в таблице по значению перв. ключа

Добавление/удаление

ADD(RECORD, TABLE) — добавить строку (запись) в таблицу и обновить все индексы
DEL(RECORD, TABLE, KEY) — удалить строку (запись) из таблицы и обновить все индексы

Прямой доступ по ключу

GET(RECORD, TABLE, KEY, KEY_VALUE) — получить строку по значению ключа
PUT(RECORD, TABLE, KEY, KEY_VALUE) — обновить строку по значению ключа
ADD(RECORD, TABLE) — добавить строку (запись) в таблицу и обновить все индексы
DEL(RECORD, TABLE, KEY) — удалить строку (запись) из таблицы и обновить все индексы

Сериальные операции в порядке индексов

```
SET(TABLE, KEY, KEY_VALUE) — установить курсор доступа в порядке индекса
NEXT(TABLE, KEY)           — просмотр таблицы в направлении с начала к концу
PREV(TABLE, KEY)           — просмотр таблицы в направлении с конца к началу
```