ЛЕКЦИЯ №5. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ. СТРУКТУРА РЕЛЯЦИОННЫХ БАЗ ДАННЫХ. ОТНОШЕНИЯ В БАЗЕ ДАННЫХ. ЭЛЕМЕНТЫ РЕЛЯЦИОННОЙ АЛГЕБРЫ И РЕЛЯЦИОННОГО ИСЧИСЛЕНИЯ.

5.1 Реляционная модель данных.

Теоретической основой этой модели стала теория отношений, основу которой заложили два логика — американец Чарльз Содерс Пирс (1839-1914) и немец Эрнст Шредер (1841-1902). В руководствах по теории отношений было показано, что множество отношений замкнуто относительно некоторых специальных операций, то есть образует вместе с этими операциями абстрактную алгебру. Это важнейшее свойство отношений было использовано в реляционной модели для разработки языка манипулирования данными, связанного с исходной алгеброй.

Будучи математиком по образованию Э.Кодд предложил использовать для обработки данных аппарат теории множеств (объединение, пересечение, разность, декартово произведение). Он показал, что любое представление данных сводится к совокупности двумерных таблиц особого вида, известного в математике как отношение – relation.

Предложения Кодда были настолько эффективны для систем баз данных, что за эту модель он был удостоен престижной премии Тьюринга в области теоретических основ вычислительной техники.

В настоящее время основным предметом критики реляционных СУБД является не их недостаточная эффективность, а следующие недостатки:

1. присущая этим системам некоторая ограниченность (прямое следствие простоты) при использовании в так называемых нетрадиционных областях (наиболее распространенными примерами являются системы автоматизации проектирования), в которых требуются предельно сложные структуры данных.
2. невозможность адекватного отражения семантики предметной области. Другими словами, возможности представления знаний о семантической специфике предметной области в реляционных системах очень ограничены. Современные исследования в области постреляционных систем главным образом посвящены именно устранению этих недостатков.

Наименьшая единица данных реляционной модели – это отдельное атомарное (неразложимое) для данной модели значение данных. Так, в одной предметной области фамилия, имя и отчество могут рассматриваться как единое значение, а в другой – как три различных значения.

Доменом называется множество атомарных значений одного и того же типа. Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа. Например, домен "Имена" в нашем примере определен на базовом типе строк символов, но в число его значений могут входить только те строки, которые могут изображать имя (в частности, такие строки не могут начинаться с мягкого знака).

Следует отметить также семантическую нагрузку понятия домена: данные считаются сравнимыми только в том случае, когда они относятся к одному домену. В нашем примере значения доменов "Номера пропусков" и "Номера групп" относятся к типу целых чисел, но не являются сравнимыми. Заметим, что в большинстве реляционных СУБД понятие домена не используется, хотя в Oracle V.7 оно уже поддерживается.

Вхождение домена в отношение принято называть ***атрибутом***.

Схема отношения – это именованное множество пар {имя атрибута, имя домена (или типа, если понятие домена не поддерживается)}

Кортеж, соответствующий данной схеме отношения, – это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. "Значение" является допустимым значением домена данного атрибута (или типа данных, если понятие домена не поддерживается). Тем самым, степень или "арность" кортежа, т.е. число элементов в нем, совпадает с "арностью" соответствующей схемы отношения. Попросту говоря, кортеж - это набор именованных значений заданного типа.

Ключ (key) — это группа из одного или более атрибутов, которая уникальным образом идентифицирует строку. Рассмотрим отношение СЕКЦИЯ (рис. 5.3), имеющее атрибуты НомерСтудента, Секция и Плата. Строка этого отношения содержит информацию о том, что студент посещает определенную секцию за определенную плату.

Отношение – это множество кортежей, соответствующих одной схеме отношения. Иногда, чтобы не путаться, говорят "отношение-схема" и "отношение-экземпляр", иногда схему отношения называют заголовком отношения, а отношение как набор кортежей - телом отношения. На самом деле, понятие схемы отношения ближе всего к понятию структурного типа данных в языках программирования. Было бы вполне логично разрешать отдельно определять схему отношения, а затем одно или несколько отношений с данной схемой.

Обычным житейским представлением отношения является двумерная таблица. Каждая строка в таблице содержит данные, относящиеся к некоторой вещи или какой-то ее части. Каждый столбец таблицы описывает какой-либо атрибут этой вещи. Иногда строки называются кортежами (tuples), а столбцы — атрибутами (attributes).

Вышеупомянутые и некоторые другие математические понятия явились теоретической базой для создания реляционных СУБД, разработки соответствующих языковых средств и программных систем, обеспечивающих их высокую производительность, и создания основ теории проектирования баз данных. Однако для массового пользователя реляционных СУБД можно с успехом использовать неформальные эквиваленты этих понятий:

- Отношение – Таблица (иногда Файл);
- Кортеж – Строка (иногда Запись);
- Атрибут – Столбец, Поле.

Реляционная база данных – это совокупность отношений, содержащих всю информацию, которая должна храниться в БД. Однако пользователи могут воспринимать такую базу данных как совокупность таблиц.

Чтобы таблица была отношением, она должна удовлетворять определенным ограничениям:

1. Во-первых, значения в ячейках таблицы должны быть одиночными — ни повторяющиеся группы, ни массивы не допускаются
2. Строки имеют фиксированное число полей (столбцов) и значений (множественные поля и повторяющиеся группы недопустимы). Иначе говоря, в каждой позиции таблицы на пересечении строки и столбца всегда имеется в точности одно значение или ничего.
3. Строки таблицы обязательно отличаются друг от друга хотя бы единственным значением, что позволяет однозначно идентифицировать любую строку такой таблицы.
4. Все записи в столбце должны быть одного типа Например, если третий столбец первой строки таблицы содержит номер сотрудника, то и во всех остальных строках таблицы третий столбец также должен содержать номер сотрудника. Каждый столбец имеет уникальное имя; порядок столбцов в таблице несуществен. Наконец, в отношении не может быть двух одинаковых строк, и порядок строк не имеет значения.
5. Полное информационное содержание базы данных представляется в виде явных значений данных и такой метод представления является единственным.
6. При выполнении операций с таблицей ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их информационному содержанию. Этому способствует наличие имен таблиц и их столбцов, а также возможность выделения любой их строки или любого набора строк с указанными признаками (например, рейсов с пунктом назначения "Париж" и временем прибытия до 12 часов).

Предложив реляционную модель данных, Э.Ф.Кодд создал и инструмент для удобной работы с отношениями – реляционную алгебру. Каждая операция этой алгебры использует одну или несколько таблиц (отношений) в качестве ее операндов и продуцирует в результате новую таблицу, т.е. позволяет "разрезать" или "склеивать" таблицы (рис. 1).

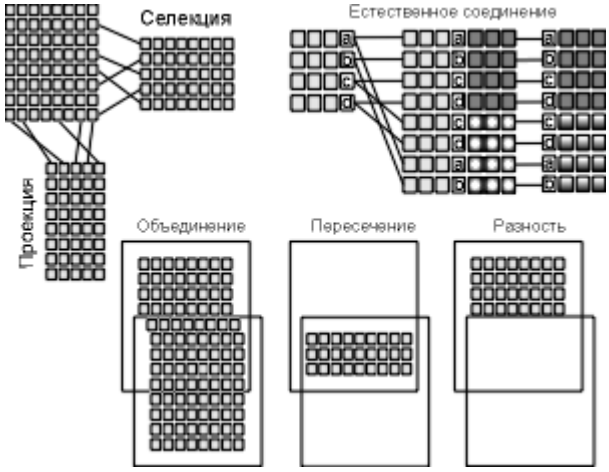

Рисунок 1 Некоторые операции реляционной алгебры.

На рис. 2 представлен пример отношения. Отношение имеет семь строк, в каждой из которых четыре столбца. Если бы мы расположили столбцы в ином порядке (скажем, поместив Табельный Номер в крайний левый столбец) или переставили бы строки (например, по возрастанию значения столбца Возраст), мы получили бы эквивалентное отношение.

	Атрибут 1 **Имя**	Атрибут 2 **Возраст**	Атрибут 3 **Пол**	Атрибут 4 **ТабельныйНомер**
Кортеж 1	Андерсон	21	Ж	010110
Кортеж 2	Деккер	22	М	010100
.	Гловер	22	М	101000
.	Джексон	21	Ж	201100
.	Мур	19	М	111100
.	Наката	20	Ж	111101
Кортеж 7	Смит	19	М	111111

Рисунок 2 Пример отношения «Сотрудник».

Созданы языки манипулирования данными, позволяющие реализовать все операции реляционной алгебры и практически любые их сочетания. Среди них наиболее распространены SQL (Structured Query Language – структуризованный язык запросов) и QBE (Quere-By-Example – запросы по образцу) [3, 5]. Оба относятся к языкам очень высокого уровня, с помощью которых пользователь указывает, какие данные необходимо получить, не уточняя процедуру их получения.

С помощью единственного запроса на любом из этих языков можно соединить несколько таблиц во временную таблицу и вырезать из нее требуемые строки и столбцы (селекция и проекция).

5.2 Получение реляционной схемы из ER-схемы

Шаг 1. Каждая простая сущность превращается в таблицу. Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

Шаг 2. Каждый атрибут становится возможным столбцом с тем же именем; может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут.

Шаг 3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификатора, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

Шаг 4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами. Т.е. делается копия уникального идентификатора с конца связи "один", и соответствующие столбцы составляют внешний ключ. Необязательные связи соответствуют столбцам, допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения.

Шаг 5. Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Шаг 6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа:

- все подтипы в одной таблице (а)
- для каждого подтипа - отдельная таблица (б)

При применении способа (а) таблица создается для наиболее внешнего супертипа, а для подтипов могут создаваться представления. В таблицу добавляется по крайней мере один столбец, содержащий код ТИПА; он становится частью первичного ключа.

При использовании метода (б) для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).

Все в одной таблице	**Таблица - на подтип**
Преимущества	
Все хранится вместе Легкий доступ к супертипу и подтипам Требуется меньше таблиц	Более ясны правила подтипов Программы работают только с нужными таблицами
Недостатки	
Слишком общее решение Требуется дополнительная логика работы с разными наборами столбцов и разными ограничениями Потенциальное узкое место (в связи с блокировками) Столбцы подтипов должны быть необязательными В некоторых СУБД для хранения неопределенных значений требуется дополнительная память	Слишком много таблиц Смущающие столбцы в представлении UNION Потенциальная потеря производительности при работе через UNION Над супертипом невозможны модификации

Шаг 7. Имеется два способа работы при наличии исключающих связей:

- общий домен (а)
- явные внешние ключи (б)

Если остающиеся внешние ключи все в одном домене, т.е. имеют общий формат (способ (а)), то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

5.3 Теоретико-множественные операции реляционной алгебры

Реляционная алгебра в том виде, в котором она была определена Эдгаром Коддом, состоит из восьми операторов, составляющих две группы по четыре оператора.

1. Традиционные операции над множествами: *объединение*, *пересечение*, *разность* и *декартово произведение* (все они модифицированы с учетом того, что их операндами являются отношения, а не произвольные множества).
2. Специальные реляционные операции: *выборка*, *проекция*, *соединение* и *деление*.

Объединение (union) – возвращает отношение, содержащее все кортежи, которые принадлежат либо одному из двух заданных отношений, либо им обоим

Пересечение (intersect) – возвращает отношение, содержащее все кортежи, которые принадлежат одновременно двум заданным отношениям

Разность (minus) – возвращает отношение, содержащее все кортежи, которые принадлежат первому из двух заданных отношений и не принадлежат второму

Произведение (times) – возвращает отношение, содержащее все возможные кортежи, которые являются сочетанием двух кортежей, принадлежащих соответственно двум заданным отношениям

Выборка – возвращает отношение, содержащее все кортежи из заданного отношения, которые удовлетворяют указанным условиям. Операцию выборки также иногда называют операцией ограничения, поэтому далее в этой книге будет также употребляться термин ограничение, если подразумевается данная алгебраическая операция

Проекция – Возвращает отношение, содержащее все кортежи (подкортежи) заданного отношения, которые остались в этом отношении после исключения из него некоторых атрибутов

Соединение – *в*озвращает отношение, содержащее все возможные кортежи, которые представляют собой комбинацию атрибутов двух кортежей, принадлежащих двум заданным отношениям, при условии, что в этих двух комбинируемых кортежах присутствуют одинаковые значения в одном или нескольких общих для исходных отношений атрибутах (причем эти общие значения в результирующем кортеже появляются один раз, а не дважды)

Произведение – Возвращает отношение, содержащее все возможные кортежи, которые являются сочетанием двух кортежей, принадлежащих соответственно двум заданным отношениям

5.3.1 Свойства реляционных операций.

Пусть A, B и C – произвольные реляционные выражения (дающие совместимые по типу отношения). Тогда для операции объединения:

- **(A UNION B) UNION C ≡ A UNION (B UNION C)** – (свойство ассоциативности)
- **A UNION B ≡ B UNION A –** (свойство коммутативности).

Аналогично свойства ассоциативности и коммутативности определяются для остальных операций.

5.4 Реляционное исчисление

Реляционное исчисление является прикладной ветвью формального механизма исчисления предикатов первого порядка. Базисными понятиями исчисления являются понятие переменной с определенной для нее областью допустимых значений и понятие правильно построенной формулы, опирающейся на переменные, предикаты и кванторы.

5.4.1 Разница между реляционной алгеброй и реляционным исчислением

Реляционная алгебра в явном виде предоставляет набор операций (соединение, объединение, проекция и т.д.), которые можно использовать, чтобы сообщить системе, как в базе данных из определенных отношений *построить* некоторое требуемое отношение, а реляционное исчисление просто представляет систему обозначений для *определения* требуемого отношения в терминах данных отношений. Например, рассмотрим запрос "Выбрать номера поставщиков и названия городов, в которых находятся поставщики детали с номером 'P2".

Алгебраическая версия этого запроса:

1. Сначала выполнить соединение отношения поставщиков S и отношения поставок SP по атрибуту S#.
2. Далее выбрать из результата этого соединения кортежи с номером детали 'P2'.
3. И наконец выполнить для результата этой выборки операцию проекции по атрибутам S# и CITY.

Этот же запрос в терминах реляционного исчисления: Получить атрибуты Si и CITY для таких поставщиков, для которых в отношении SP существует запись о поставке с тем же значением атрибута S# и со значением атрибута P#, равным 'P2'.

На самом деле *реляционная алгебра и реляционное исчисление* логически эквивалентны. Каждому выражению в алгебре соответствует эквивалентное выражение в исчислении, и точно так каждому выражению в исчислении соответствует эквивалентное выражение в алгебре. Это означает, что между ними существует взаимнооднозначное соответствие, а различия связаны лишь с разными стилями выражения: *исчисление* ближе к естественному языку, а *алгебра* — к языку программирования.