4.2. Реляционная модель данных

Реляционная модель впервые была представлена широким массам специалистов по моделированию данных Эдгаром Ф. Коддом (Edgar Frank Codd) в 1970 году в его основополагающей статье «Реляционная модель данных для больших совместно используемых банков данных». Публикацию этой статьи принято считать поворотным пунктом в истории развития систем баз данных. Цели создания реляционной модели формулировались следующим образом.

- Обеспечение более высокой степени независимости программ от данных. Прикладные программы не должны зависеть от изменений внутреннего представления данных, в частности от изменений организации файлов, переупорядочивания записей и путей доступа.
- Создание прочного фундамента для решения семантических вопросов, а также проблем непротиворечивости и избыточности данных. В частности, в статье Кодда вводится понятие нормализованных отношений, т.е. отношений с атомарными значениями атрибутов в кортежах.
- Расширение языков управления данными за счет включения операций над множествами (спецификационных операций).

Хотя достоинства реляционной модели привлекли внимание и возбудили интерес к ней многих людей, все же наиболее значительные исследования были проведены в рамках трех проектов. Первый из них разрабатывался в конце 1970-х годов в исследовательской лаборатории корпорации IBM в городе Сан-Хосе, штат Калифорния, под руководством Астрахана (Astrahan), результатом чего стало создание системы под названием System R, являвшейся прототипом истинной реляционной СУБД. Этот проект был задуман с целью получения реальных доказательств практической применимости реляционной модели посредством реализации предусматриваемых ею структур данных и операций. Этот проект также зарекомендовал себя как важнейший источник информации о таких проблемах реализации, как управление транзакциями, управление параллельной работой, технология восстановления, оптимизация запросов, обеспечение безопасности и целостности данных, учет человеческого фактора и разработка пользовательского интерфейса. Выполнение проекта стимулировало публикацию многих научно-исследовательских статей и создание других прототипов реляционных СУБД. В частности, работа над проектом System R дала толчок проведению следующих важнейших разработок:

- создание языка структурированных запросов SQL, который с тех пор приобрел статус формального стандарта ISO (International Organization for Standardization) и в настоящее время является фактическим стандартом языка реляционных СУБД;
- создание различных коммерческих реляционных СУБД, которые впервые появились на рынке в конце 1970-х и начале 1980-х годов, таких как DB2 корпорации IBM и Oracle корпорации Oracle Corporation.

Вторым проектом, который сыграл заметную роль в разработке реляционной модели данных, был проект INGRES (INteractive Graphics REtrieval System), работа над которым проводилась в Калифорнийском университете (город Беркли) почти в то же самое время, что и над проектом System R. Проект INGRES включал разработку прототипа реляционной СУБД с концентрацией основного внимания на тех же общих целях, что и проект System R. Эти исследования привели к появлению академической версии INGRES, которая внесла существенный вклад в общее признание реляционной модели данных. Позже от данного проекта отпочковались коммерческие продукты INGRES фирмы Relational Technology Inc. (теперь INGRES II фирмы Computer Associates) и Intelligent Database Machine фирмы Britton Lee Inc.

Третьим проектом была система Peterlee Relational Test Vehicle научного центра корпорации IBM, расположенного в городе Петерли, Великобритания. Этот проект был более теоретическим, чем проекты System R и INGRES. Его результаты имели очень большое и даже принципиальное значение, особенно в таких областях, как обработка запросов и оптимизация, а также функциональные расширения системы.

К числу наибольших достоинств реляционного подхода можно отнести:

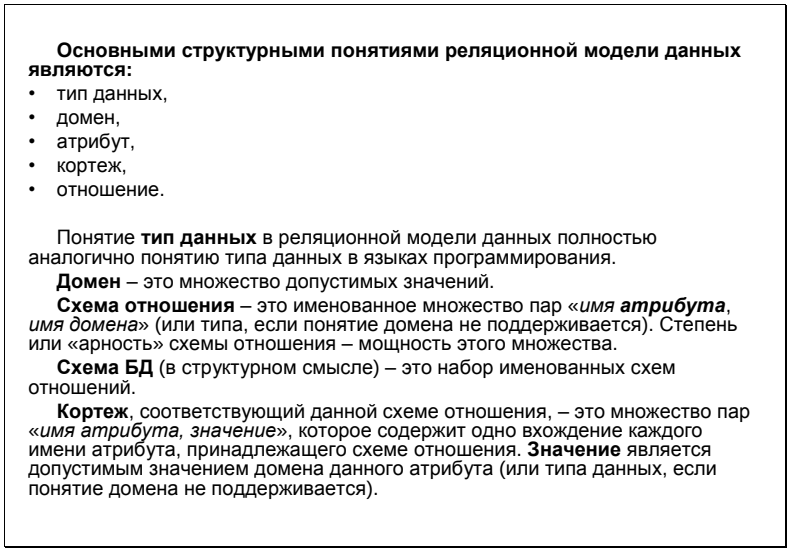
- наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными;
- наличие простого и в то же время мощного математического аппарата, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации баз данных;
- возможность ненавигационного (спецификационного) манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

Реляционные системы далеко не сразу получили широкое распространение. В то время как основные теоретические результаты в этой области были получены еще в 70-х годах, и тогда же появились первые прототипы реляционных СУБД, долгое время считалось невозможным добиться эффективной реализации таких систем. Однако отмеченные выше преимущества и постепенное накопление методов и алгоритмов организации реляционных баз данных и управления ими привели к тому, что уже в середине 80-х годов реляционные системы практически вытеснили с мирового рынка ранние СУБД.

В настоящее время основным предметом критики реляционных СУБД является не их недостаточная эффективность, а присущая этим системам некоторая ограниченность (прямое следствие простоты) при использовании в так называемых нетрадиционных областях применения (наиболее распространенными примерами являются системы автоматизации проектирования), в которых требуются предельно сложные структуры данных. Еще одним часто отмечаемым недостатком реляционных баз данных является 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 (ООСУБД и ОРСУБД, упоминавшихся в предыдущем параграфе) главным образом посвящены именно устранению этих недостатков.

Коммерческие системы на основе реляционной модели данных начали появляться в конце 1970-х - начале 1980-х годов. В настоящее время существует несколько сотен типов различных реляционных СУБД, как для мэйнфреймов, так и для персональных компьютеров, хотя многие из них не полностью соответствуют точному определению реляционной модели данных. Примерами реляционных СУБД для персональных компьютеров (кроме уже упомянутых DB2 и Oracle) являются СУБД MS SQL, Access и FoxPro фирмы Microsoft, Paradox фирмы Corel Corporation, InterBase фирмы Borland, свободно распространяемые СУБД MySQL и PostgreSQL.

4.2.1. Структуры



Основными структурными понятиями реляционной модели данных являются:

- тип данных,
- домен,
- атрибут,
- кортеж,
- отношение.

Эти уже знакомые вам термины обсуждались во второй главе именно с позиций реляционной модели, поскольку именно в ней эти формы данных нашли свое истинное воплощение.

Понятие **тип данных** в реляционной модели данных полностью аналогично понятию типа данных в языках программирования. Обычно в современных реляционных БД допускается хранение символьных, числовых данных, битовых строк, специализированных числовых данных (таких как «*деньги*»), а также специальных «темпоральных» данных (*дата*, *время*, *временной интервал*). Достаточно активно развивается подход к расширению возможностей реляционных систем абстрактными типами данных (соответствующими возможностями обладают, например, системы семейства Ingres/Postgres).

Понятие домена более специфично для баз данных, хотя и имеет некоторые аналогии с подтипами в некоторых языках программирования. В самом общем виде **домен** определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат «*истина*», то элемент данных является элементом домена. Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа. Например, домен «*Имена*» может быть 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. Значения доменов «*Номера палат*» и «*Регистрационные номера пациентов*» относятся к типу целых чисел, но не являются сравнимыми. Заметим, что в большинстве реляционных СУБД понятие домена не используется.

Определение 4.2.1.1. Схема отношения – это именованное множество пар «*имя **атрибута**, имя домена*» (или типа, если понятие домена не поддерживается). Степень или «арность» схемы отношения – мощность этого множества.

Определение 4.2.1.2. Схема БД (в структурном смысле) – это набор именованных схем отношений.

Определение 4.2.1.3. 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 заданных типов.

Определение 4.2.1.4. 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. Однако в реляционных базах данных это не принято. Имя схемы отношения в таких базах данных всегда совпадает с именем соответствующего отношения-экземпляра.

Определение 4.2.1.5. Реляционная база данных – это набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

В классических реляционных базах данных после определения схемы базы данных изменяются только отношения-экземпляры. В них могут появляться новые и удаляться или модифицироваться существующие кортежи. Однако во многих реализациях допускается и изменение схемы базы данных: определение новых и изменение существующих схем отношений. Это принято называть эволюцией схемы базы данных.

Обычным житейским представлением отношения является **таблица, заголовком** которой является схема отношения, а **строками** – кортежи отношения-экземпляра; в этом случае имена атрибутов именуют столбцы этой таблицы. Поэтому иногда говорят «**столбец** таблицы», имея в виду «атрибут отношения». Этой терминологии придерживаются в большинстве коммерческих реляционных СУБД.

Как видно, основные структурные понятия реляционной модели данных имеют очень простую интуитивную интерпретацию, хотя в теории реляционных БД все они определяются абсолютно формально и точно.

Термин **отношение** в общепринятом математическом смысле определяется следующим образом. Для заданных множеств S_1, S_2, ..., S_n (не обязательно различных) R является отношением на этих n множествах, если представляет собой множество кортежей степени n, у каждого из которых первый элемент взят из множества S_1, второй – из множества S_2 и т.д. Мы будем называть S_j j-тым **доменом** R. Говорят, что такое отношение R имеет **степень** n. Отношения степени 1 часто называют **унарными**, степени 2 – **бинарными**, степени 3 – **тернарными**, степени 4 – **кватернарными** и степени n – ***n*-арными**.

n-арное «математическое» отношение R обладает следующими свойствами:

- Порядок кортежей не является существенным.
- Все кортежи различны.
- Порядок значений атрибутов в кортежах является существенным – он соответствует порядку доменов, на которых определяется R.

В своей основополагающей статье Кодд определяет структурные понятия своей модели следующим образом. Термин отношение используется в его общепринятом математическом смысле.

Определение 4.2.1.6. Для заданных множеств S_1, S_2, ..., S_n (не обязательно различных) R является **отношением** на этих n множествах, если представляет собой множество кортежей степени n, у каждого из которых первый элемент взят из множества S_1, второй – из множества S_2 и т.д. Мы будем называть S_j j-тым **доменом** R. Говорят, что такое отношение R имеет **степень** n. Отношения степени 1 часто называют **унарными**, степени 2 – **бинарными**, степени 3 – **тернарными**, степени 4 – **кватернарными** и степени n – ***n*-арными.**

Для наглядности часто используется представление отношений в виде массивов (таблиц), но нужно помнить, что это конкретное представление не является существенной частью реляционного представления. Массив, представляющий n-арное отношение R, обладает следующими свойствами.

- Каждая строка представляет кортеж степени n.
- Порядок строк не является существенным.
- Все строки различны.
- Порядок столбцов является существенным – он соответствует порядку доменов, на которых определяется R.

Термин **отношение** в моделировании данных определяется следующим образом.
Пусть задано множество из *n* типов или доменов $T_i (i = 1,..., n)$, причем все они необязательно должны быть различными. Тогда *r* будет отношением, определенным на этих типах, если оно состоит из двух частей: заголовка и тела (заголовок еще иногда называют схемой или интенсионалом отношения, а тело – экстенсионалом отношения), где:

- заголовок – это множество из *n* атрибутов вида $A_i : T_i$;

здесь A_i – имена атрибутов отношения *r*, а T_i – соответствующие имена типов;

- тело – это множество из *m* кортежей *t* ; здесь *t* является множеством компонентов вида $A_i : v_i$, в которых v_i – значение типа T_i , т.е. значение атрибута A_i в кортеже *t*.

Отношения реляционной модели обладают следующими свойствами:

- отсутствие кортежей-дубликатов,
- отсутствие упорядоченности кортежей,
- отсутствие упорядоченности атрибутов,
- атомарность значений атрибутов.

Позже «математический» взгляд на отношения, при котором операция Декартова произведения не коммутативна, был заменен на представление отношения в смысле определения Дейта (определение 2.2.15), которое мы дали во второй главе. В результате отношения реляционной модели стали обладать следующими свойствами.

Отсутствие кортежей-дубликатов

То свойство, что отношения не содержат кортежей-дубликатов, следует из определения отношения как множества кортежей. В классической теории множеств, по определению, каждое множество состоит из различных элементов. Из этого свойства вытекает наличие у каждого отношения так называемого первичного ключа – набора атрибутов, значения которых однозначно определяют кортеж отношения. Для каждого отношения, по крайней мере, полный набор его атрибутов обладает этим свойством. Однако при формальном определении первичного ключа требуется обеспечение его «минимальности», т.е. в набор атрибутов первичного ключа не должны входить такие атрибуты, которые можно отбросить без ущерба для основного свойства, – однозначно определять кортеж. Понятие первичного ключа является исключительно важным в связи с понятием целостности баз данных.

Заметим, что во многих практических реализациях РСУБД допускается нарушение свойства уникальности кортежей отношения. Такие отношения являются не множествами, а мультимножествами, что в ряде случаев позволяет добиться определенных преимуществ, но иногда приводит к серьезным проблемам.

Отсутствие упорядоченности кортежей

Свойство отсутствия упорядоченности кортежей отношения также является следствием определения отношения-экземпляра как множества кортежей. Отсутствие требования к поддержанию порядка на множестве кортежей отношения дает дополнительную гибкость СУБД при хранении баз данных во внешней памяти и при выполнении запросов к базе данных. Это не противоречит тому, что при формулировании запроса к БД, например, на языке SQL можно потребовать сортировки результирующей таблицы в соответствии со значениями некоторых столбцов. Такой результат, это вообще говоря, не отношение, а некоторый упорядоченный список кортежей.

Отсутствие упорядоченности атрибутов

Атрибуты отношений не упорядочены, поскольку по определению схема отношения есть множество пар «*имя атрибута*, *имя домена*». Для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута. Это свойство теоретически позволяет, например, модифицировать схемы существующих отношений не только путем добавления новых атрибутов, но и путем удаления существующих атрибутов. Однако в большинстве существующих систем такая возможность не

допускается, и хотя упорядоченность набора атрибутов отношения явно не требуется, часто в качестве неявного порядка атрибутов используется их порядок в линейной форме определения схемы отношения.

Атомарность значений атрибутов

Значения всех атрибутов являются атомарными. Это следует из определения домена как потенциального множества значений простого типа данных, т.е. среди значений домена не могут содержаться множества значений и агрегаты значений. Принято говорить, что в реляционных базах данных допускаются только нормализованные отношения или отношения, представленные в первой нормальной форме.

Реляционная схема медицинской ПрО (структуры)

БОЛЬНИЦА (Код больницы (К/Б), Название, Адрес, Число коек (Ч/К))
ПАЛАТА (Код больницы (К/Б), Номер палаты (Н/П), Название, Число коек (Ч/К))
ПЕРСОНАЛ (Код больницы (К/Б), Номер палаты (Н/П), Фамилия, Должность, Смена, Зарплата (З/П))
ВРАЧ (Код врача (К/В), Код больницы (К/Б), Фамилия, Специальность)
ПАЦИЕНТ (Регистрационный номер (Р/Н), Фамилия, Адрес, Дата рождения (Д/Р), Пол, Номер медицинского полиса (НМП))
ДИАГНОЗ (Регистрационный номер (Р/Н), Тип диагноза (Т/Д), Осложнения, Предупреждающая информация)
ЛАБОРАТОРИЯ (Код лаборатории (К/Л), Название, Адрес)
АНАЛИЗ (Регистрационный номер (Р/Н), Код Лаборатории (К/Л), Тип анализа (Т/А), Назначенная дата (Н/Д), Назначенное время (Н/В), Номер направления (Н/Н), Состояние)
БОЛЬНИЦА-ЛАБОРАТОРИЯ (Код больницы (К/Б), Код Лаборатории (К/Л))
ВРАЧ-ПАЦИЕНТ (Код врача (К/В), Регистрационный номер (Р/Н))
РАЗМЕЩЕНИЕ (Код больницы (К/Б), Номер палаты (Н/П), Регистрационный номер (Р/Н), Номер койки (Н/К))
ТЕЛЕФОН БОЛЬНИЦЫ (Код больницы (К/Б), Телефон)
ТЕЛЕФОН ЛАБОРАТОРИИ (Код лаборатории (К/Л), Телефон)

На слайде представлена реляционная схема медицинской ПрО, построенная для ER-схемы этой ПрО, приведенной в конце пункта 3.2.2. Люди, склонные к поиску закономерностей, на основании сравнения этих схем без труда могут синтезировать простейшие правила преобразования схем.

Простейшие правила перехода от ER-схемы к реляционной схеме БД

1. **Каждое множество сущностей представляется самостоятельным отношением, однозначные атрибуты множества сущностей становятся атрибутами отношения, ключи множества сущностей являются возможными ключами отношения; при необходимости в качестве первичного ключа отношения используется суррогатный атрибут.**
2. **Бинарные множества связей типа 1:M без атрибутов представляются дублированием первичного ключа 1-отношения в M-отношение.**
3. **Бинарные множества связей типа M:N без атрибутов представляются самостоятельными отношениями, куда дублируются первичные ключи отношений, построенных для множеств сущностей.**
4. **Множества связей с атрибутами представляются самостоятельными отношениями, куда дублируются первичные ключи отношений, построенных для множеств сущностей. Однозначные атрибуты множества связей становятся атрибутами этого отношения.**

Приведем эти правила трансформации схем ПрО из ER-модели Чена в реляционную модель.

1. Каждое множество сущностей представляется самостоятельным отношением, однозначные атрибуты множества сущностей становятся атрибутами отношения, ключи множества сущностей являются возможными ключами отношения; при необходимости в качестве первичного ключа отношения используется суррогатный атрибут.

Применяя это правило, мы получили первые восемь отношений нашей схемы. Обратите внимание на отличия в составах атрибутов множеств сущностей и отношений. Во-первых, в эти отношения не попали многозначные атрибуты множеств сущностей. Во-вторых, в некоторые отношения добавлены суррогатные первичные ключи (о них мы поговорим подробнее в следующем пункте).

2. Бинарные множества связей типа 1:1 или 1:*M* без атрибутов представляются дублированием первичного ключа 1-отношения в *M*-отношение.

1-отношение – это отношение, построенное для представления в реляционной модели множества сущностей, возле которого на ребре множества связей стояла пометка «1» в ER-диаграмме. Соответственно, *M*-отношение – это отношение, построенное для множества сущностей, возле которого на ребре множества связей стояла пометка «*M*». В случае множества связей типа 1:1 выбор *M*-отношения осуществляется произвольным образом. В соответствии с этим правилом добавлены следующие атрибуты: *Код больницы* – в отношения *ПАЛАТА* и *ВРАЧ*, *Регистрационный номер* – в отношения *ДИАГНОЗ* и *АНАЛИЗ*, *Код лаборатории* – в отношение *АНАЛИЗ*, *Код больницы*, *Номер палаты* – в отношение *ПЕРСОНАЛ*.

3. Бинарные множества связей типа *M*:*N* без атрибутов представляются самостоятельными отношениями, куда дублируются первичные ключи отношений, построенных для множеств сущностей.

По этому правилу построены отношения *БОЛЬНИЦА-ЛАБОРАТОРИЯ* и *ВРАЧ-ПАЦИЕНТ*.

4. Множества связей с атрибутами представляются самостоятельными отношениями, куда дублируются первичные ключи отношений, построенных для множеств сущностей. Однозначные атрибуты множества связей становятся атрибутами этого отношения.

Этому правилу обязано своим появлением в реляционной схеме отношение *РАЗМЕЩЕНИЕ*. У нас было единственное множество связей такого рода – *РАЗМЕЩЕНИЕ*, имеющее однозначный атрибут *Номер койки*.

Простейшие правила перехода от ER-схемы к реляционной схеме БД (продолжение)

5. **Множества связей степени больше 2-х представляются самостоятельными отношениями, куда дублируются первичные ключи отношений, построенных для множеств сущностей. Однозначные атрибуты множества связей становятся атрибутами этого отношения.**
6. **Каждый многозначный атрибут множества сущностей представляется отдельным отношением, куда дублируется первичный ключ отношения, построенного для множества сущностей; второй атрибут этого отношения предназначен собственно для значения.**
7. **Каждый многозначный атрибут множества связей представляется отдельным отношением, куда дублируется первичный ключ отношения, построенного для множества связей; второй атрибут этого отношения предназначен собственно для значения.**

5. Множества связей степени больше 2-х представляются самостоятельными отношениями, куда дублируются первичные ключи отношений, построенных для множеств сущностей. Однозначные атрибуты множества связей становятся атрибутами этого отношения.

В нашей ER-схеме не было множеств связей степени больше двух. Применение этого правила схоже с третьим правилом. Единственное отличие заключается в количестве атрибутов нового отношения. Для третьего правила оно всегда равно двум, а для пятого правила – степени исходного множества связей (возможные однозначные атрибуты множества связей в обоих случаях в расчет не принимаются).

6. Каждый многозначный атрибут множества сущностей представляется отдельным отношением, куда дублируется первичный ключ отношения, построенного для множества сущностей; второй атрибут этого отношения предназначен собственно для значения.

Это правило породило отношения *ТЕЛЕФОН БОЛЬНИЦЫ* и *ТЕЛЕФОН ЛАБОРАТОРИИ* для хранения значений многозначных атрибутов *ТЕЛЕФОН* множеств сущностей *БОЛЬНИЦА* и *ЛАБОРАТОРИЯ* соответственно.

7. Каждый многозначный атрибут множества связей представляется отдельным отношением, куда дублируется первичный ключ отношения, построенного для множества связей; второй атрибут этого отношения предназначен собственно для значения.

В нашей ER-схеме не было многозначных атрибутов у множеств связей.

Эффективность результирующей реляционной схемы БД существенным образом зависит от выразительной мощности используемой семантической модели данных и полноты набора правил трансформации схем. Только что мы познакомились с простейшим набором этих правил для преобразования «ER-модель Чена – реляционная модель». В последней главе книги мы рассмотрим более детальный набор правил этого преобразования, а также познакомимся с идеями и правилами преобразований «EER-модель – реляционная модель» и «ERM-модель – реляционная модель».

Представления

В реляционной модели слово «представление» (англ. view) означает виртуальное или производное отношение (анл. virtual relation), т.е. отношение, экстенсионал которого на самом деле не хранится на диске, но динамически воспроизводится на основании одного или нескольких базовых отношений (отношений, реально существующих в базе данных). Пользователь может одновременно использовать как базовые отношения, так и представления.

Определение 4.2.1.7. Базовое отношение – отношение, кортежи которого физически хранятся в базе данных.

Определение 4.2.1.8. Представление – динамический результат одной или нескольких реляционных операций над базовыми отношениями с целью создания некоторого иного отношения. Представление является виртуальным отношением, которое реально в базе данных не существует, но создается по требованию пользователя в момент обращения к представлению.

С точки зрения пользователя представление является отношением, которое постоянно существует и с которым можно работать точно так же как с базовым отношением. В частности, можно определять представление не только на основе базовых отношений, но и на основе других представлений.

Содержимое представления определяется как результат выполнения запроса к одному или нескольким отношениям. Любые операции над представлением автоматически транслируются в операции над отношениями, на основании которых оно было создано. Представления имеют динамический характер, т.е. изменения в базовых отношениях, которые могут повлиять на содержимое представления, немедленно, отражаются на содержимом этого представления. Если пользователи вносят в представление некоторые допустимые изменения, последние немедленно заносятся в базовые отношения представления.

Представления используются по нескольким причинам.

• Они предоставляют мощный и гибкий механизм защиты, позволяющий скрыть некоторые части базы данных от определенных пользователей. Пользователь не будет иметь сведений о существовании атрибутов или кортежей, отсутствующих в доступных ему представлениях.

• Они позволяют организовать доступ пользователей к данным наиболее удобным для них образом, поэтому одни и те же данные в одно и то же время могут рассматриваться разными пользователями совершенно различными способами.

• Они позволяют упрощать сложные операции с базовыми отношениями. Например, если представление будет определено на основе соединения двух отношений, то пользователь сможет выполнять над ним простые унарные операции выборки и проекции, которые будут автоматически преобразованы средствами СУБД в эквивалентные операции с выполнением соединения базовых отношений.

Однако на самом деле представления позволяют добиться и более важного типа логической независимости от данных, связанной с защитой пользователей от реорганизаций схемы. Например, если в отношение будет добавлен новый атрибут, то пользователи не будут даже подозревать о его существовании, пока определения их представлений не включают этот атрибут. Если существующее отношение реорганизовано или разбито на части, то использующее его представление может быть переопределено так, чтобы пользователи могли продолжать работать с данными в прежнем формате.

Все обновления данных в базовом отношении должны быть немедленно отражены во всех представлениях, связанных с этим базовым отношением. Аналогично, при обновлении данных в представлении внесенные изменения должны быть отражены в его базовом отношении. Но на типы изменений, которые могут быть выполнены с помощью

представлений, накладываются определенные ограничения. Ниже перечислены условия, которые используются в большинстве существующих систем для проверки допустимости обновления данных через некоторое представление.

• Обновления допускаются через представление, которое определено на основе простого запроса к единственному базовому отношению и содержит первичный или потенциальный ключ этого базового отношения.

• Обновления не допускаются в любых представлениях, определенных на основе нескольких базовых отношений.

• Обновления не допускаются в любых представлениях, включающих выполнение операций агрегирования или группирования.

4.2.2. Ограничения целостности

Конструкции языка SQL, связанные с ограничениями целостности:

- **тип данных атрибута;**
- **описатели атрибута NULL и NOT NULL, позволяющие определить соответственно, может или нет атрибут иметь неопределенное значение в кортежах отношения;**
- **описатель атрибута или группы атрибутов PRIMARY KEY, определяющий первичный ключ отношения;**
- **описатель атрибута или группы атрибутов UNIQUE, определяющий возможный ключ отношения;**
- **конструкция FOREIGN KEY (REFERENCES), определяющая внешний ключ отношения;**
- **конструкция CHECK, определяющая условие на значения атрибута(-ов), которому должны удовлетворять все кортежи отношения.**

В предыдущем пункте была рассмотрена структурная часть реляционной модели данных. В этом пункте рассматриваются реляционные ограничения целостности, а в следующих двух – реляционные операции управления данными.

Начиная с этого момента, мы будем знакомить читателей с языками определения данных и языками манипулирования данными реляционной модели. Говоря о средствах определения схемы РБД, конечно нельзя обойти вниманием повсеместно используемый язык SQL (об истории создания языка и его командах манипулирования данными мы расскажем в пункте 4.2.4).

Определение реляционной схемы в среде SQL осуществляется выполнением команд *CREATE TABLE* (первоначальное создание схемы отношения) и *ALTER TABLE* (последующая модификация схемы отношения). Всю полноту работы со схемой отношения обеспечивает команда *CREATE TABLE*, именно поэтому ее мы подробно рассмотрим в этом пункте. В частности, на слайде приведены средства задания ограничений целостности команды *CREATE TABLE*. Но до того, как вводить грамматику этой команды, познакомимся с основными понятиями ограничений целостности реляционной модели.

Поскольку каждый атрибут связан с некоторым доменом, для множества допустимых значений каждого атрибута отношения определяются так называемые **ограничения домена**. Помимо этого, задаются два важных правила целостности, которые, по сути, являются ограничениями для всех допустимых состояний базы данных. Эти два основных правила реляционной модели иногда называют целостностью сущностей и ссылочной целостностью. Однако прежде чем приступить к изучению этих правил, следует рассмотреть понятие пустых значений.

Определение 4.2.2.1. Неопределенное значение (англ. NULL) указывает, что значение атрибута в настоящий момент неизвестно или неприемлемо для этого кортежа. Ключевое слово *NULL* представляет собой способ обработки неполных или необычных данных.

Неопределенное значение не принадлежит никакому типу данных и может присутствовать среди значений любого атрибута, определенного на любом типе данных (если это явно не запрещено при определении атрибута). Если *a* – это значение некоторого типа данных или *NULL*, *op* – любая двуместная операция этого типа данных (например, +), а *lop* – операция сравнения значений этого типа (например, =), то по определению:

a op NULL = NULL

NULL op a = NULL
a lop NULL = unknown
NULL lop a = unknown

Здесь *unknown* – это третье значение логического, или булевского, типа, обладающее следующими свойствами:

NOT unknown = unknown
true AND unknown = unknown
true OR unknown = true
false AND unknown = false
false OR unknown = unknown.

NULL не следует понимать как нулевое численное значение или заполненную пробелами текстовую строку. Нули и пробелы представляют собой некоторые значения, тогда как ключевое слово *NULL* обозначает отсутствие какого-либо значения. Поэтому *NULL* следует рассматривать иначе, чем другие значения. Некоторые авторы используют термин «*значение NULL*», но на самом деле *NULL* не является значением, а лишь обозначает его отсутствие, поэтому термин «*значение NULL*» можно использовать лишь с определенными оговорками.

Использование понятия *NULL* в реляционной модели является спорным вопросом. Кодд рассматривает понятие *NULL* как составную часть этой модели, а другие специалисты считают этот подход неправильным, полагая, что проблема отсутствующей информации еще не до конца понята, удовлетворительное ее решение не найдено, а потому включение определителя *NULL* в реляционную модель является преждевременным.

В целостной части реляционной модели данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД. Первое требование называется **требованием целостности сущностей**. Объекту или сущности реального мира в реляционных БД соответствует кортеж отношения. Конкретно, требование состоит в том, что любой кортеж любого отношения должен быть отличим от любого другого кортежа этого отношения, т.е. другими словами, любое отношение должно обладать **первичным ключом**. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

Второе требование называется **требованием целостности по ссылкам**. Очевидно, что при соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений. Для определения соответствия между кортежами отношений значения первичного ключа одного отношения дублируются в специальном атрибуте второго (возможно, того же самого) отношения. Атрибут такого рода называется **внешним ключом**, поскольку его значения однозначно характеризуют сущности, представленные кортежами другого отношения. Говорят, что отношение, в котором определен внешний ключ, ссылается на соответствующее отношение, в котором такой же атрибут является первичным ключом. Требование целостности по ссылкам, или требование внешнего ключа, состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть полностью неопределенным (т.е. ни на что не указывать).

Для соблюдения целостности сущности достаточно гарантировать отсутствие отношений, содержащих кортежи с одним и тем же значением первичного ключа и запрещать вхождение в значение первичного ключа неопределенных значений.

С целостностью по ссылкам дела обстоят несколько более сложно. Понятно, что при обновлении ссылающегося отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа. Но как быть при удалении

кортежа из отношения, на которое ведет ссылка? Здесь существуют три подхода, каждый из которых поддерживает целостность по ссылкам. Первый подход заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить их значения внешнего ключа). При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи. В развитых реляционных СУБД обычно можно выбрать способ поддержания целостности по ссылкам для каждой отдельной ситуации определения внешнего ключа. Конечно, для принятия такого решения необходимо анализировать требования конкретной прикладной области.

Как указано выше, в отношении не должно быть повторяющихся кортежей. Поэтому необходимо иметь возможность уникальной идентификации каждого отдельного кортежа отношения по значениям одного или нескольких атрибутов (называемых **реляционными ключами**).

Определение 4.2.2.2. Суперключ (англ. superkey). Атрибут или множество атрибутов, которое единственным образом идентифицирует кортеж данного отношения.

Суперключ однозначно обозначает каждый кортеж в отношении. Но суперключ может содержать дополнительные атрибуты, которые необязательны для уникальной идентификации кортежа, поэтому нас будут интересовать суперключи, состоящие только из тех атрибутов, которые действительно необходимы для уникальной идентификации кортежей.

Определение 4.2.2.3. Потенциальный ключ (англ. candidate key). Суперключ, который не содержит подмножества, также являющегося суперключом данного отношения.

Потенциальный ключ K для данного отношения R обладает двумя свойствами.

• Уникальность. В каждом кортеже отношения R значение ключа K единственным образом идентифицируют этот кортеж.

• Неприводимость. Никакое допустимое подмножество ключа K не обладает свойством уникальности.

Отношение может иметь несколько потенциальных ключей. Если ключ состоит из нескольких атрибутов, то он называется **составным ключом**

Обратите внимание на то, что любой конкретный набор кортежей отношения нельзя использовать для доказательства того, что некий атрибут или комбинация атрибутов являются потенциальным ключом. Тот факт, что в некоторый момент времени не существует значений-дубликатов, совсем не означает, что их не может быть вообще. Однако наличие значений-дубликатов в конкретном существующем наборе кортежей вполне может быть использовано для демонстрации того, что некоторая комбинация атрибутов не может быть потенциальным ключом. Для идентификации потенциального ключа требуется знать смысл используемых атрибутов в «реальном мире»; только это позволит обоснованно принять решение о возможности существования значений-дубликатов. Только, исходя из подобной семантической информации, можно гарантировать, что некоторая комбинация атрибутов является потенциальным ключом отношения.

Определение 4.2.2.4. Первичный ключ (англ. primary key). Потенциальный ключ, который выбран для уникальной идентификации кортежей внутри отношения. Поскольку отношение не содержит кортежей-дубликатов, всегда можно уникальным образом идентифицировать каждую его строку. Это значит, что отношение всегда имеет первичный ключ. В худшем случае все множество атрибутов может использоваться как первичный ключ, но обычно, чтобы различить кортежи, достаточно использовать

несколько меньшее подмножество атрибутов. Потенциальные ключи, которые не выбраны в качестве первичного ключа, называются **альтернативными ключами.**

Одним из основных принципов выбора первичного ключа из множества потенциальных ключей отношения является семантический принцип. Коль скоро по значению первичного ключа всегда уникально идентифицируется кортеж отношения, все атрибуты первичного ключа не должны содержать неопределенного значения ни в одном из кортежей отношения. Второй принцип заботится скорее об эффективности работы с данными и предлагает выбирать в качестве первичного ключа такой потенциальный ключ, в котором количество атрибутов и суммарная длина значений минимальны.

В тех случаях, когда у отношения нет компактного первичного ключа, часто прибегают к использованию суррогатных ключей. Хотя многие профессиональные проектировщики давно используют в качестве первичных ключей всех отношений БД исключительно их.

Определение 4.2.2.5. Суррогатный ключ (англ. surrogate key) – это искусственный атрибут отношения, не имеющий связей с какой-либо естественной характеристикой явлений ПрО и вводимый в схему отношения исключительно для организации связей кортежей этого отношения с кортежами других отношений. Его основное назначение – обеспечивать уникальную внутрисистемную идентификацию кортежей отношения, в котором он объявлен как первичный ключ. Для этого СУБД обеспечивает механизм генерации уникальных значений этого атрибута для вновь создаваемых кортежей.

По сравнению с естественными первичными ключами у суррогатных ключей есть следующие преимущества:

- единообразие всех суррогатных ключей отношений схемы БД (обычно их имя составляется из стандартного префикса или суффикса и имени отношения – *Код сотрудника* или *СТУДЕНТ_ID,* и все они имеют один и тот же тип значений – *INTEGER* со стандартной максимальной длиной);
- компактность суррогатных ключей;
- отсутствие потребности в изменениях, которые могут затронуть много отношений схемы (обычно такое возможно в случае ошибочно введенных значений естественных первичных ключей, которые сдублированы в другие отношения для представления связей между кортежами).

Определение 4.2.2.6. Внешний ключ (англ. foreign key)**.** Атрибут или множество атрибутов внутри отношения, который соответствует потенциальному (как правило, первичному) ключу некоторого (может быть, того же самого) отношения и является его подмножеством. Если некий атрибут присутствует в нескольких отношениях, то его наличие обычно отражает определенную связь между кортежами этих отношений. Как будет показано далее, эти общие атрибуты играют важную роль в манипулировании данными.

В подавляющем большинстве случаев связей между кортежами по внешнему ключу используется правило равенства: кортеж, в котором имеется определенное значение внешнего ключа, связан с другим кортежем в том случае, если значение внешнего ключа совпадает со значением первичного ключа во втором кортеже. Причем тот факт, что во втором отношении это значение уникально (требование целостности сущностей), означает гарантированную функциональность соответствующего отображения между этими отношениями.

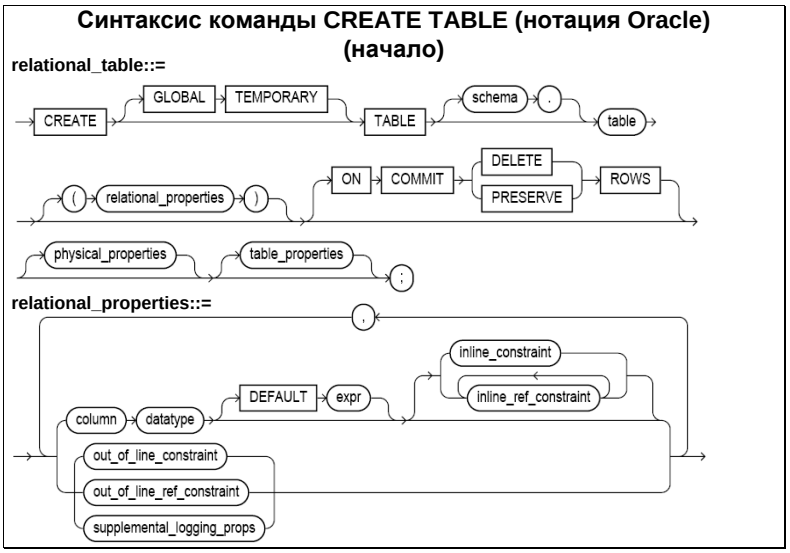
Целостность сущностей. В базовом отношении ни один атрибут первичного ключа не может содержать отсутствующих значений, обозначаемых как *NULL*. По определению, первичный ключ – это минимальный идентификатор, который используется для уникальной идентификации кортежей. Это значит, что никакое подмножество первичного ключа не может быть достаточным для уникальной идентификации кортежей. Если допустить присутствие *NULL* в любой части первичного ключа, это равносильно

утверждению, что не все его атрибуты необходимы для уникальной идентификации кортежей, что противоречит определению первичного ключа.

Ссылочная целостность. Если в отношении существует внешний ключ, то значение внешнего ключа должно либо соответствовать значению потенциального (как правило, первичного) ключа некоторого кортежа в его базовом отношении либо внешний ключ должен полностью состоять из значений *NULL*.

Корпоративные ограничения целостности – дополнительные правила поддержки целостности данных, определяемые пользователями или администраторами базы данных. Пользователи сами могут указывать дополнительные ограничения, которым должны удовлетворять данные. Например, если в одном отделении не может работать больше 20 сотрудников, то пользователь может указать это как правило, а СУБД должна следить за его выполнением. В этом случае в отношение нельзя будет добавить строку со сведениями о новом сотруднике некоторого отделения, если в данном отделении компании уже насчитывается 20 сотрудников.

Декларативная поддержка реляционной целостности в разных системах отличается. Мы рассмотрим основные функциональные возможности диалекта языка SQL, реализованного в СУБД Oracle. Как уже упоминалось, эта СУБД поддерживает объектно-реляционную модель. Поскольку нас интересует реализация в ней реляционной модели, все конструкции команд SQL, касающиеся объектных расширений, будут оставлены без внимания. Также мы умолчим о конструкциях, имеющих отношение к физической модели этой СУБД и позволяющих оптимизировать хранение и использование данных без изменения их логической схемы.



Рассказ о командах SQL будет сопровождаться приведением на слайдах грамматики этих команд. При этом будет использоваться диаграммная нотация, взятая из документации по СУБД Oracle. Эта нотация раскрывает синтаксис команд в виде правил.

В левой части правила (перед знаком «::=») указан нетерминальный символ (нетерминал), синтаксис которого приведен в правой части правила. Все нетерминальные символы составляют строчные буквы английского алфавита. Если в правой части встречается нетерминальный символ, в конкретной команде он должен быть заменен на конструкцию, для которой либо есть соответствующее синтаксическое правило, либо приводится пояснение в тексте комментариев.

Все терминальные символы (терминалы) составляют либо английские слова, записанные прописными буквами, либо специальные символы. Они в неизменном виде кодируются в команде.

Для того, чтобы сформировать синтаксически правильную команду, необходимо пройти по какому-то маршруту в графе диаграммы этой команды, начиная с ее левого конца и заканчивая на ее правом конце. Если по пути будет встречен нетерминал, для которого предусмотрено отдельное правило, надо как бы включить граф этого правила вместо нетерминала.

Полный синтаксис команды определения схемы одного отношения соответствует нетерминальному символу «*relational_table*».

Поясним некоторые элементы грамматики команды *CREATE TABLE*.

GLOBAL TEMPORARY – глобальная временная таблица, доступная всем сеансам; каждый сеанс видит только те данные, которые созданы в нем.

schema – имя схемы, в которой создается таблица (по умолчанию используется схема пользователя, в сеансе которого выполняется команда).

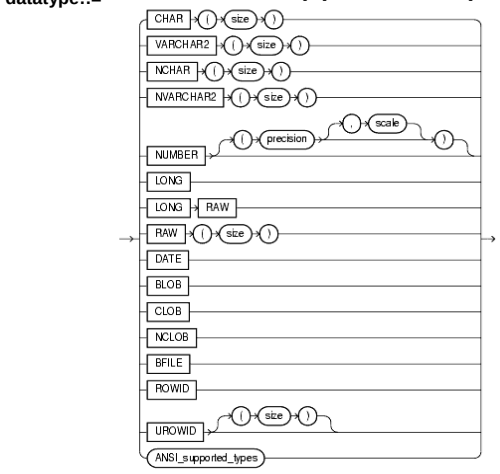
table – имя создаваемой таблицы.

ON COMMIT-конструкция используется только для глобальных временных таблиц и указывает, доступны ли ее данные только во время транзакции или во время сеанса.

column – имя столбца.

DEFAULT-конструкция задает значение столбца по умолчанию.

Синтаксис команды CREATE TABLE (нотация Oracle) (продолжение)



CHAR (size) – строки символов стандартного алфавита фиксированной длины (правые пробелы хранятся); *size* – максимальная длина (от 1 до 2000 байт).

VARCHAR2 (size) – строки символов стандартного алфавита переменной длины (правые пробелы не хранятся); *size* – максимальная длина (от 1 до 4000 байт).

NCHAR (size) – строки символов дополнительного национального алфавита фиксированной длины (правые пробелы хранятся); *size* – максимальная длина (от 1 до 2000 байт).

NVARCHAR2 (size) – строки символов дополнительного национального алфавита переменной длины (правые пробелы не хранятся); *size* – максимальная длина (от 1 до 4000 байт).

NUMBER [(precision, [scale])] – числа; *precision* – общее число цифр (от 1 до 38), *scale* – число цифр после десятичной точки (от -84 до 127).

LONG – символьные данные с переменной длиной до 2 гигабайт.

LONG RAW – двоичные данные с переменной длиной до 2 гигабайт.

RAW (size) – двоичные данные с фиксированной длиной; *size* – длина (от 1 до 2000 байт).

DATE – даты и время в интервале от 1 января 4712 года до нашей эры до 31 декабря 9999 нашей эры.

BLOB – большие двоичные данные; максимальный размер – 4 гигабайта.

CLOB – большие символьные данные (из 1-байтных символов стандартного алфавита); максимальный размер – 4 гигабайта.

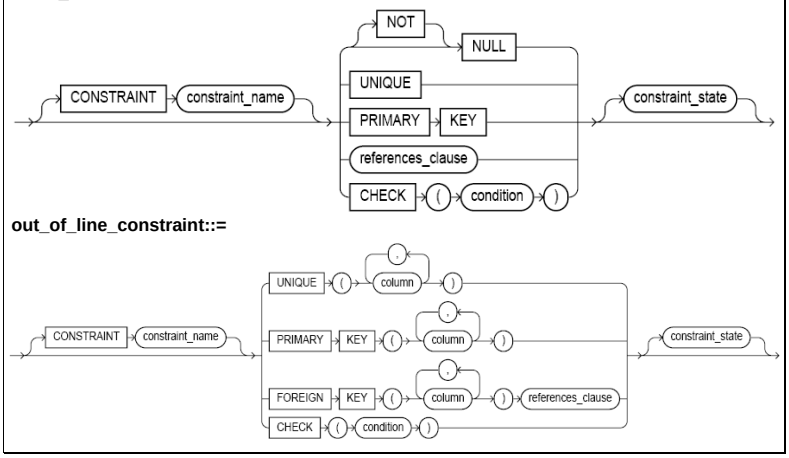
NCLOB – большие символьные данные (из символов дополнительного национального алфавита); максимальный размер – 4 гигабайта.

BFILE – путь к большому двоичному файлу, хранящемуся вне БД Oracle в операционной системе.

ROWID – уникальный адрес строки таблицы в базе данных Oracle.

UROWID [(size)] – универсальный логический адрес строки таблицы в базе данных под управлением других СУБД.

Синтаксис команды CREATE TABLE (нотация Oracle) (продолжение)



inline_constraint определяет ограничение целостности на уровне столбца, *out_of_line_constraint* – на уровне таблицы (позволяет использовать группы столбцов).

constraint_name – уникальное имя ограничения целостности (если опущено, используется системное имя).

NOT NULL запрещает столбцу принимать значение *NULL*.

NULL (по умолчанию) разрешает столбцу принимать значение *NULL*.

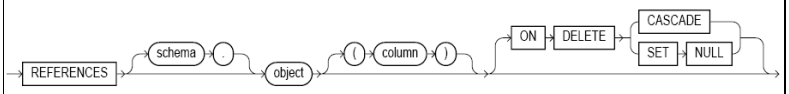
UNIQUE определяет потенциальный ключ.

PRIMARY KEY определяет первичный ключ.

CHECK-конструкция определяет условие (*condition*), которому должны удовлетворять все строки таблицы. Условие должно соответствовать синтаксису условий команды *SELECT* (с большими ограничениями). Оно представляет собой логическое выражение, определенное с точностью до имен столбцов (*inline_constraint* может содержать имя одного столбца, *out_of_line_constraint* – может ссылаться на несколько столбцов таблицы). В момент проверки ограничения целостности для строки ее значения столбцов подставляются в выражение, и оно оценивается.

Синтаксис команды CREATE TABLE (нотация Oracle) (продолжение)

references_clause::=



references_clause определяет ограничение ссылочной целостности для внешнего ключа таблицы.

schema – имя схемы, которой принадлежит таблица, на первичный или потенциальный ключ которой ссылается определяемый внешний ключ.

object – имя таблицы, на первичный или потенциальный ключ которой ссылается определяемый внешний ключ.

column – имя одного или нескольких столбцов (через запятую) таблицы *object*, составляющих ее первичный или потенциальный ключ.

ON DELETE-конструкция определяет поведение *Oracle* при удалении значения первичного или потенциального ключа в таблице, на которую ссылается внешний ключ. Если конструкция опущена, *Oracle* не разрешит сделать это, если такое значение используется во внешнем ключе. Если указано *CASCADE*, автоматически удалятся строки ссылающейся таблицы с удаляемым значением. Если указано *SET NULL*, удаляемые значения в строках ссылающейся таблицы будут заменены на *NULL*.

На слайде приведены примеры команд CREATE TABLE для отношений реляционной схемы медицинской БД.

Процедурный способ определения ограничений целостности с помощью триггеров

Определение 4.2.2.7. Триггер (англ. trigger) – это программа на языке программирования сервера (для Oracle – это PL/SQL), которая автоматически выполняется СУБД в момент наступления определенного события. Триггер подобен процедуре PL/SQL тем, что представляет собой именованный блок PL/SQL с разделами объявления, выполнения и обработки исключительных ситуаций. Однако, в отличие от обычной процедуры, триггер выполняется неявно в каждом случае возникновения триггерного события, к тому же не имеет параметров. Приведение триггера в действие иногда называют запуском (англ. firing), или активизацией триггера. Триггеры могут использоваться для достижения следующих целей:

- проверка правильности введенных данных и проверка выполнения сложных ограничений целостности данных, которые трудно, если вообще возможно, поддерживать с помощью декларативных ограничений целостности, установленных для таблицы;
- осуществление косвенных модификаций данных, сопутствующих действиям пользователей (например, в случае денормализации схемы);
- выдача предупреждений (например, с помощью электронной почты), которые напоминают о необходимости выполнить некоторые действия;
- накопление информации аудита посредством фиксации сведений о внесенных изменениях и тех лицах, которые их выполнили.

Триггер определяет действие, которое должно быть предпринято базой данных при возникновении в приложении некоторого события. Триггеры базируются на модели "событие-условие-действие" (англ. Event-Condition-Action – ECA).

Как события, которые управляют триггерами в Oracle, рассматриваются:

- операторы *INSERT, UPDATE* или *DELETE*, применяемые к указанной таблице (или, возможно, представлению);
- операторы *CREATE, ALTER* или *DROP*, применяемые к любому объекту схемы;
- запуск базы данных или останов экземпляра Oracle;
- регистрация пользователя в системе или выход из нее;
- конкретное или любое сообщение об ошибке.

Также для триггера можно определить:

- когда должен сработать триггер – до или после события;
- условие, которое определяет, должно ли быть выполнено действие (условие является необязательным, но если оно определено, то действие должно быть выполнено только тогда, когда это условие истинно);
- действие, которое должно быть предпринято (этот блок содержит операторы PL/SQL, которые должны быть выполнены, когда выдается активизирующий оператор и условие активизации триггера принимает истинное значение).

Есть два типа триггеров: строковые триггеры (англ. row-level triggers), которые выполняются для каждой затронутой активизирующим событием строки таблицы, и операторные триггеры (англ. statement-level triggers), выполняющиеся только один раз для всего события, даже если активизирующее событие затрагивает множество строк. База данных Oracle поддерживает также триггеры *INSTEAD OF*, обеспечивающие прозрачный способ модификации представлений, которые не могут быть модифицированы непосредственно с помощью операторов SQL (*INSERT, UPDATE* и *DELETE*). Эти триггеры называются триггерами *INSTEAD OF*, поскольку, в отличие от триггеров других

типов, база данных Oracle активизирует их вместо (англ. instead of) выполнения первоначального оператора SQL.

Триггеры могут также активизировать друг друга. Это может происходить, когда действие триггера влечет за собой внесение изменения в базе данных, которое, в свою очередь, вызывает другое событие, с которым также связан триггер.

Ключевое слово *NEW* в теле триггера используется для ссылки на новое значение столбца, ключевое слово *OLD* может быть использовано для ссылки на старое значение столбца. Ясно, что ссылки на старые значения не применимы для событий вставки, а на новые – для событий удаления. Синтаксис обращений – {:*NEW* | :*OLD*}.<имя столбца>.

Триггерные события включают вставку, удаление и обновление строк в таблице. Но только в последнем случае для триггерного события можно также указать конкретные имена столбцов таблицы. Время запуска триггера определяется с помощью ключевых слов *BEFORE* и *AFTER*: при указании ключевого слова *BEFORE* триггер запускается до возникновения связанных с ним событий, а при указании ключевого слова *AFTER* – после их возникновения. Выполняемые триггером действия задаются программой на PL/SQL, которая может быть выполнена одним из двух следующих способов:

- для каждой строки (*FOR EACH ROW*), охваченной данным событием (так называемый триггер на уровне строки);
- только один раз для всей команды (если *FOR EACH ROW* отсутствует).

Тело триггера не может содержать:

- операторы SQL, применяемые при обработке транзакций, например *COMMIT* и *ROLLBACK*;
- операторы SQL, которые служат для установления соединения, например *CONNECT* и *DISCONNECT*;
- операторы SQL для определения схемы и управления ею, например команды создания или удаления таблиц;
- операторы SQL для управления сеансом, например *SET SESSION CHARACTERISTICS, SET ROLE, SET TIME ZONE*.

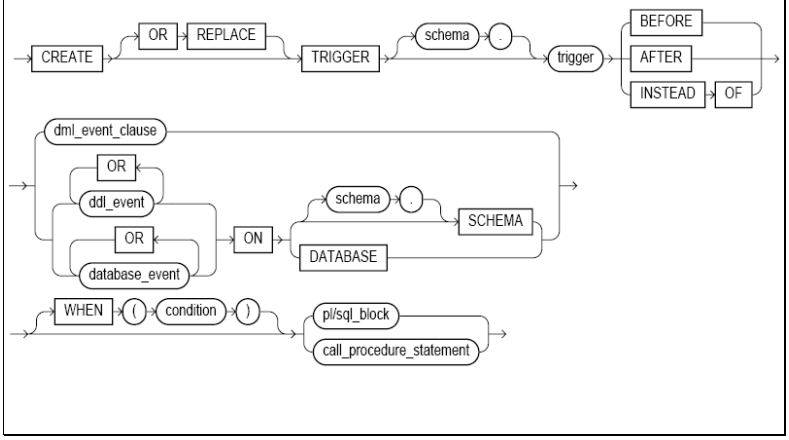
Кроме того, стандарт SQL не предусматривает возможности создания динамически изменяющихся триггеров (т.е. триггеров, позволяющих при обработке события вносить изменения в код самого триггера, а затем вызывать его снова в цикле, который может стать бесконечным). Поскольку для таблицы может быть определено несколько триггеров, то большое значение имеет порядок их запуска.

Запуск триггеров происходит по мере возникновения триггерных событий (*INSERT, UPDATE, DELETE*) в следующем порядке:

1. Исполнение табличного триггера *BEFORE* на уровне оператора.
2. Для каждой строки, охваченной данным оператором:
   а) исполнение любого триггера *BEFORE* на уровне строки;
   б) блокировка данных и выполнение над ними действия, предписанного командой SQL;
   в) выполнение проверок ограничений целостности;
   г) исполнение любого триггера *AFTER* на уровне строки.
3. Исполнение табличного триггера *AFTER* на уровне оператора.

Следует отметить, что при использовании такой последовательности запуска триггеры *BEFORE* активизируются перед проверкой ограничений целостности, поэтому нельзя исключить такую возможность, что внесение изменения, вызвавшего активизацию триггера, приведет к нарушению ограничений целостности базы данных и поэтому должно быть запрещено. Таким образом, при разработке триггеров *BEFORE* необходимо руководствоваться правилом, что они не должны вносить дополнительных изменений в базу данных.

Синтаксис команды CREATE TRIGGER (нотация Oracle) (начало)



На слайде представлен синтаксис команды *CREATE TRIGGER* в SQL-диалекте Oracle. Хотя, как уже отмечалось, триггеры можно определять для команд языка определения схемы и системных событий, происходящих в СУБД, мы ограничимся рассмотрением триггеров для команд языка манипулирования данными, наиболее часто используемых при разработке систем БД.

Поясним некоторые элементы грамматики команды *CREATE TRIGGER*.

schema – имя схемы, в которой создается триггер (по умолчанию используется схема пользователя, в сеансе которого выполняется команда).

trigger – имя триггера.

BEFORE указывает Oracle выполнять триггер до выполнения основного действия. Строковый триггер с этой опцией будет вызываться перед выполнением основного действия для каждой строки. Ограничения: нельзя использовать в триггерах для представлений; в теле триггера можно изменять *NEW*-значения и нельзя изменять *OLD*-значения столбцов.

AFTER указывает Oracle выполнять триггер после выполнения основного действия. Строковый триггер с этой опцией будет вызываться после выполнения основного действия для каждой строки. Ограничения: нельзя использовать в триггерах для представлений; в теле триггера нельзя изменять ни *NEW*-значения, ни *OLD*-значения столбцов.

INSTEAD OF указывает Oracle выполнять триггер вместо выполнения основного действия. Такие триггеры допустимы только для событий модификации данных представлений. Ограничения: нельзя использовать в триггерах для таблиц; в теле триггера можно читать *NEW*-значения и *OLD*-значения и нельзя изменять ни *NEW*-значения, ни *OLD*-значения столбцов.

Замечание. Можно создать несколько триггеров одного и того же типа на одно и то же событие. Порядок их выполнения не определен. Если вашему приложению требуется выполнять действия триггеров в определенном порядке, скомпонуйте их в виде одного триггера.

dml_event_clause определяет грамматику триггеров для команд языка манипулирования данными.

ddl_event определяет грамматику триггеров для команд языка определения данных.

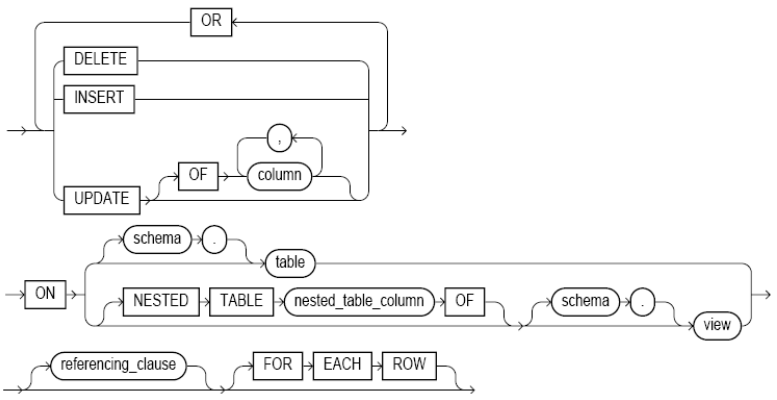
database_event определяет грамматику триггеров для системных событий.

WHEN определяет дополнительное условие, при истинности которого будет выполняться триггер.

pl/sql_block определяет блок PL/SQL, который выполняется, когда триггер активизируется.

Синтаксис команды CREATE TRIGGER (нотация Oracle) (продолжение)

dml_event_clause::=



DELETE указывает, что триггер будет выполняться для действия удаления строк таблицы.

INSERT указывает, что триггер будет выполняться для действия добавления строк в таблицу.

UPDATE указывает, что триггер будет выполняться для действия модификации строк таблицы. После ключевого слова *OF* можно указать список имен столбцов, только при изменений значений которых будет вызываться триггер.

ON определяет объект базы данных (таблицу или представление), для команд модификации данных которого создается триггер.

FOR EACH ROW определяет является ли триггер строковым. Если эта фраза опущена, триггер является операторным.

При условии правильного использования триггеры могут стать очень мощным механизмом. Основное преимущество триггеров заключается в том, что с их помощью стандартные функции могут сохраняться внутри базы данных и неизменно активизироваться при каждом обновлении ее данных. Это позволяет существенно упростить приложение.

Тем не менее, следует упомянуть и о присущих триггерам недостатках.

- Сложность. При перемещении из приложения в базу данных некоторых функций усложняются задачи ее проектирования, реализации и администрирования.
- Скрытые функциональные средства. Перемещение некоторых функций в базу данных и сохранение их в виде одного или нескольких триггеров может привести к сокрытию от пользователя некоторых функциональных возможностей. Хотя это в определенной степени упрощает работу пользователя, но, к сожалению, может привести к появлению незапланированных, потенциально нежелательных и вредных побочных эффектов, поскольку в этом случае пользователь не может контролировать некоторые процессы, происходящие в базе данных.
- Влияние на производительность. Перед выполнением каждого оператора по изменению состояния базы данных СУБД должна проверить триггерное условие с целью выяснения необходимости запуска триггера для этого оператора. Данная операция сказывается на общей производительности СУБД. Очевидно, что при возрастании количества триггеров накладные расходы, связанные с такими действиями, также возрастают. В моменты

пиковой нагрузки они могут вызвать заметное снижение производительности системы.

4.2.3. Навигационные операции

Навигационный стиль манипулирования реляционными данными используется исключительно в программных интерфейсах с реляционными СУБД. Все эти интерфейсы строятся по одному и тому же принципу.

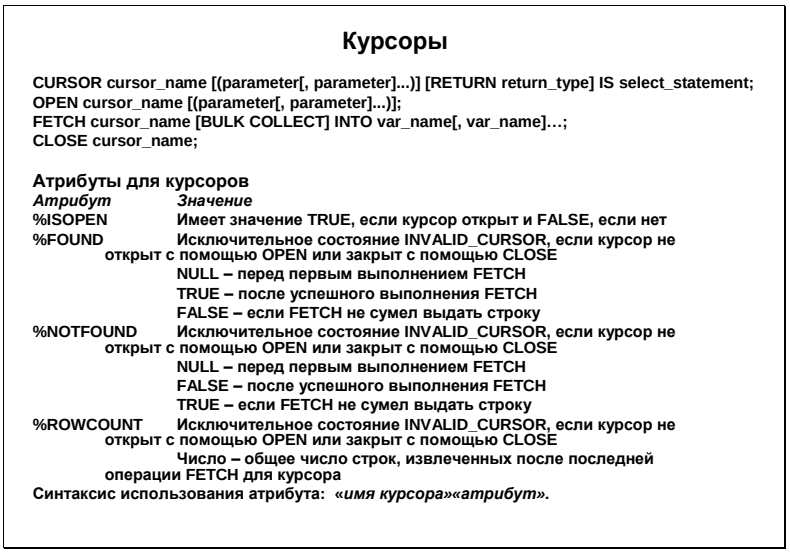
СУБД передается спецификационный запрос на выборку данных, которому в общем случае удовлетворяет несколько строк. В пространстве памяти программы отводится место для размещения читаемых из текущей строки значений. Далее с помощью операторов цикла языка программирования организуется построчное сканирование результирующей выборки с чтением и последующей обработкой данных одной строки. Для обеспечения произвольных алгоритмов манипулирования данными в навигационном языке должна быть предусмотрена возможность параллельного независимого сканирования одной и той же или различных таблиц.

По такому принципу строятся все интерфейсы между прикладными программами на различных языках программирования и сервером БД с участием специализированных или универсальных драйверов, таких как ODBC, OLE DB или ADO. Отличия заключаются в терминологии и способах представления встроенных инструментов в программе.

Мы познакомимся с навигационным языком манипулирования данными на примере механизма курсоров языка PL/SQL, основного и до недавнего времени единственного языка программирования сервера Oracle.

PL/SQL-программы в виде процедур, функций, триггеров хранятся и выполняются на сервере и потому обеспечивают единообразное эффективное выполнение сложных алгоритмов преобразования данных, процедурной проверки ограничений целостности, которые не могут быть представлены в декларативной форме в команде *CREATE TABLE*. Являясь в каком-то смысле процедурным расширением языка SQL, PL/SQL обеспечивает мощный и в то же время простой механизм выполнения всех команд языка манипулирования данными SQL, включая управление транзакциями.

Аппарат курсоров языка PL/SQL позволяет добавить к функциональности спецификационных команд SQL возможность построчного сканирования результирующих таблиц с данными.



PL/SQL использует два типа курсоров – неявные и явные. Первые организуются системой неявно для всех команд SQL, если они селектируют и выполняют те или иные действия с одной строкой. Однако для запросов, которые возвращают несколько строк, в PL/SQL необходимо объявить явный курсор, открыть его, в цикле этого курсора просканировать строки результата, выполняя требуемые действия с данными, и закрыть курсор.

Прежде чем использовать курсор его необходимо объявить в области объявлений (*DECLARE*) PL/SQL-программы. Объявление курсора начинается с ключевого слова *CURSOR* и имеет синтаксис, показанный на слайде. Обязательными элементами этого предложения являются имя курсора (*cursor_name*) и связанный с курсором запрос в виде команды *SELECT* (*select_statement*). Необязательные элементы – параметры (*parameter*) и возвращаемый тип (*return_type*) – позволяют определять курсор с точностью до значений параметров и манипулировать целиком результирующей строкой, а не отдельными значениями столбцов, соответственно.

Команды *OPEN* и *CLOSE* представляют собой своеобразные операторные скобки, внутри которых курсор активен и может использоваться для сканирования данных. При открытии курсора командой *OPEN* (синтаксис приведен на слайде) по указанному имени курсора определяется команда *SELECT*, связанная с этим курсором, она выполняется, и формируется результирующая таблица. Первоначально указатель курсора устанавливается в начало этой таблицы на ее первую строку.

Если в команде *SELECT* указано предложение *FOR UPDATE*, в момент открытия курсора блокируются все строки таблицы-результата. Это предотвращает любые изменения этих данных в БД другими пользователями.

Если курсор объявлен с одним или несколькими формальными параметрами, их фактические значения можно указать в команде *OPEN*. Значение параметра можно не задавать, если в его определении предусмотрено значение по умолчанию. Для задания соответствия между актуальными значениями и формальными параметрами курсора можно использовать как позиционный, так и именованный способ. Типы данных значения и параметра должны быть совместимыми.

Чтение строк командой *OPEN* не производится, для этого предназначена команда *FETCH*.

Если вы не используете конструкцию *BULK COLLECT*, каждое обращение к команде *FETCH* читает данные из текущей строки результирующей таблицы и перемещает указатель курсора на следующую строку. Для каждого столбца результирующей таблицы должна быть предусмотрена своя переменная, совместимая по

типу данных с этим столбцом. Именно в них *FETCH* помещает значения текущей строки курсора. Имена этих переменных указываются после ключевого слова *INTO* в точном соответствии с целевым списком команды *SELECT* курсора.

Конструкция *BULK COLLECT* команды *FETCH* позволяет за одно обращение к ней прочитать целиком все столбцы результирующей таблицы в соответствующие коллекции, типы данных которых соответствуют типам данных столбцов.

Команда *CLOSE* закрывает курсор. После этого становятся недоступными его результирующая таблица и указатель курсора. Закрытый курсор можно повторно открыть, однако неизменность результирующей таблицы по понятным причинам не гарантирована.

Каждый явный курсор имеет четыре атрибута – %*ISOPEN*, %*FOUND*, %*NOTFOUND*, %*ROWCOUNT*. Эти атрибуты возвращают полезную информацию о выполнении обращений к данным результирующей таблицы. Их следует использовать для адекватной обработки ситуаций, связанных с курсором.

%*ISOPEN* возвращает значение *TRUE*, если курсор открыт, *FALSE* – в противном случае.

%*FOUND* принимает значение *NULL*, если курсор открыт, но команды *FETCH* для него не выполнялись, *TRUE*, если последняя команда *FETCH* прочитала данные из текущей строки и *FALSE*, если очередной *FETCH* вышел за границу курсора. До открытия курсора и после его закрытия обращение к атрибуту %*FOUND* вызывает исключительную ситуацию *INVALID_CURSOR*.

%*NOTFOUND* отличается от %*FOUND* тем, что он возвращает *TRUE*, когда %*FOUND* дает *FALSE* и наоборот. В остальных случаях реакция аналогична.

%*ROWCOUNT* равен нулю сразу после открытия курсора и числу строк результирующей таблицы, прочитанных до этого командами *FETCH* в процессе сканирования. До открытия курсора и после его закрытия обращение к атрибуту %*ROWCOUNT* вызывает исключительную ситуацию *INVALID_CURSOR*.

Обращения к атрибутам курсоров имеют следующий синтаксис – «*имя курсора*» «*атрибут*».

Курсоры (продолжение)

```
DECLARE
   p_name ПЕРСОНАЛ.Фамилия%TYPE;
   p_sal ПЕРСОНАЛ.З/П%TYPE;
   CURSOR cursor1 IS
        SELECT Фамилия, З/П FROM ПЕРСОНАЛ
        WHERE Должность = 'СИДЕЛКА';
BEGIN
…
   OPEN cursor1;
   LOOP
        FETCH cursor1 INTO p_name, p_sal;
        EXIT WHEN cursor1%NOTFOUND;
        -- обработка одной записи
   END LOOP;
   CLOSE cursor1;
…
END;
```

На слайде приведены фрагменты PL/SQL-программы, в которой построчной обработке подвергаются фамилии и зарплаты персонала в должности «*СИДЕЛКА*».

В области объявлений кроме курсора определены две скалярных переменных, типы данных которых совпадают с типами данных соответствующих столбцов таблицы *ПЕРСОНАЛ*. Обратите внимание, как это делается с помощью конструкции *%TYPE*, применяемой к столбцу таблицы.

Для организации сканирования результирующей таблицы использован безусловный цикл (*LOOP … END LOOP*). Выход из цикла обеспечивается оператором *EXIT* с условием, в котором используется атрибут курсора *%NOTFOUND*.

Вопросы и задания к пунктам 4.2.1, 4.2.2, 4.2.3

1. Каковы основные структурные понятия реляционной модели данных?
2. Перечислите простейшие правила перехода от ER-схемы к реляционной схеме БД.
3. Что такое представление (view)?
4. Перечислите конструкции языка SQL, связанные с ограничениями целостности.
5. Что такое неопределенное значение?
6. Охарактеризуйте требование целостности сущностей и требование целостности по ссылкам.
7. Дайте определения всем вариантам реляционных ключей.
8. Назовите основные компоненты команды SQL *CREATE TABLE*.
9. Что такое триггеры и для чего они предназначены?
10. Назовите основные компоненты команды SQL *CREATE TRIGGER*.
11. Каковы основные особенности навигационного стиля манипулирования реляционными данными?
12. Назовите команды аппарата курсоров языка PL/SQL. Опишите схему использования курсоров.