4.2.4. Спецификационные операции

Спецификационные языки реляционной модели:

- алгебраические языки (реляционная алгебра Кодда);

- языки исчисления предикатов
- реляционное исчисление с переменными-кортежами (ALPHA),
- реляционное исчисление с переменными на доменах (QBE);

- язык SQL.

До появления реляционной модели манипулирование данными осуществлялось исключительно с помощью навигационных операций, каждая из которых выполняла действие с единственным элементом БД. Идея спецификационных операций, позволяющих за одно обращение произвести действие сразу над многими элементами, внесена в моделирование данных Коддом как одна из основных идей реляционной модели. Он же предложил два спецификационных языка для своей модели. Позже появились другие спецификационные языки реляционной модели, а также спецификационные языки для дореляционных моделей.

Принято все спецификационные языки реляционной модели делить на следующие классы:

1. Алгебраические языки, позволяющие выражать запросы средствами специализированных операторов, применяемых к отношениям. Мы рассмотрим один из вариантов реляционной алгебры, предложенный Коддом, с традиционно упоминаемыми всеми авторами операциями над отношениями.
2. Языки исчисления предикатов, в которых запросы описывают требуемое множество кортежей путем указания логического выражения, которому должны удовлетворять эти кортежи. В зависимости от вида объектов, которые означивают переменные, языки этого класса делятся на:
   - языки реляционного исчисления с переменными-кортежами и
   - языки реляционного исчисления с переменными на доменах.

   Представителем первого подкласса является язык ALPHA, второго – язык QBE (с ними нам тоже предстоит познакомиться).
3. SQL-подобные языки, которые некоторые авторы относят к классу реляционных языков, основанных на отображениях. Действительно, каждая конструкция SELECT этих языков задает отображение строк исходных таблиц в строки результирующей таблицы.

Несмотря на то, что влиятельные международные организации давно занимаются стандартизацией языка SQL, каждая фирма, разрабатывающая СУБД, реализует в своей системе свой диалект языка. Пожалуй, лишь конструкции стандарта SQL-89 более-менее единообразно представлены во всех диалектах. Мы познакомимся с базовыми возможностями языка SQL, предложенными в стандарте SQL-89, а затем подробно разберем синтаксис и семантику команд SQL-диалекта, реализованного в СУБД Oracle.

Первые два класса языков на абстрактном уровне были предложены Коддом. В таком виде языки этих классов не реализованы в какой-либо действующей СУБД и служат эталоном для оценки мощности языков запросов существующих систем. Каждый из трех абстрактных языков (реляционной алгебры, реляционного исчисления с переменными-кортежами и реляционного исчисления с переменными на доменах) эквивалентен по своей выразительности двум другим. Собственно и предложены они были Коддом для представления минимальных возможностей любого разумного языка запросов, использующего реляционную модель.

Реальные языки запросов (которые мы также рассмотрим в соответствующих подпунктах) обычно обеспечивают не только функции абстрактных, но и некоторые дополнительные потребности пользователей.

Следует отметить, что спецификационные реляционные языки можно разделить на процедурные и декларативные. В первых языках запрос определяет, как найти результат, во вторых – что требуется найти. Поэтому в первом случае запрос представляет собой некий алгоритм получения результата, а во втором – образец или условие, которому должны соответствовать требуемые данные. Отметим, что навигационные языки всегда процедурны.

4.2.4.1. Реляционная алгебра

Реляционная алгебра Кодда

Совместимые отношения* – это отношения, у которых совпадают заголовки (количество и имена атрибутов)

Основные операции

1. **ОБЪЕДИНЕНИЕ*** $R \cup S = \{t \mid t \in R \vee t \in S\}$
2. **РАЗНОСТЬ*** $R - S = \{t \mid t \in R \wedge t \notin S\}$
3. **ДЕКАРТОВО ПРОИЗВЕДЕНИЕ** $R \times S = \{r \| s \mid r \in R \wedge s \in S\}$
4. **ПРОЕКЦИЯ**

$$\pi_{i_1,\dots,i_m}(R) = \{< a_1,\dots,a_m > \mid \exists < b_1,\dots,b_k > (< b_1,\dots,b_k > \in R \wedge \forall j = \overline{1,m}(a_j = b_{i_j}))\}$$

5. **СЕЛЕКЦИЯ** $\sigma_F(R) = \{r \mid r \in R \wedge F\}$

Дополнительные операции

1. **ПЕРЕСЕЧЕНИЕ*** $R \cap S = \{t \mid t \in R \wedge t \in S\} = R - (R - S)$
2. **ЧАСТНОЕ** $R \div S = \{t^{(r-s)} \mid \forall u^{(s)}(u \in S \to t \| u \in R)\}$

$$R \div S = \pi_{1,\dots,r-s}(R) - \pi_{1,\dots,r-s}((\pi_{1,\dots,r-s}(R) \times S) - R)$$

3. **СОЕДИНЕНИЕ** $R \overset{\triangleright\triangleleft}{}_{i\theta j} S = \sigma_{i\theta(r+j)}(R \times S)$

$$R \overset{\triangleright\triangleleft}{}_{X\theta Y} S = \{r \| s \mid r \in R \wedge s \in S \wedge (\pi_X(r) \theta \pi_Y(s))\}$$

Реляционная алгебра представляет собой набор операторов, определенных на множестве отношений (т.е. использующих отношения в качестве операндов и возвращающих отношения в качестве результата). Кодд определил так называемую «начальную» алгебру отношений – набор из восьми операторов, которые мы и рассмотрим. На самом деле можно определить любое число операторов, которые удовлетворяют условию «несколько отношений на входе – одно отношение на выходе» (многие исследователи так и делали, и поэтому существует много вариантов реляционных алгебр).

Перед определением операторов реляционной алгебры необходимо сделать ряд замечаний.

Представляется удобным обращаться к значениям в кортеже по именам соответствующих атрибутов, хотя в некоторых случаях (при математическом взгляде на отношения) целесообразно считать компоненты кортежей неименованными и обращаться к ним по порядковым номерам. При определении реляционной алгебры предполагается, что атрибуты не обязательно должны быть именованными и порядок в кортежах существен.

Операндами реляционной алгебры являются базовые и производные отношения. Степень отношения будет уточняться лишь, когда это существенно.

Теоретико-множественные операции (объединение, разность, пересечение) могут применяться только к **совместимым отношениям** – отношениям с совпадающим набором атрибутов.

Последнее замечание касается результата любых операций реляционной алгебры. Поскольку он представляет собой отношение, а отношение – это множество кортежей, в нем не должно быть кортежей-дубликатов. Поэтому последней фазой любой операции является нормализация результирующего отношения – удаление из него кортежей-дубликатов. Особенно следует помнить об этом при выполнении таких операций, как объединение и проекция.

Для определения реляционной алгебры используются пять основных операций. После их представления мы приведем несколько дополнительных операций, которые не расширяют функциональности языка, однако обеспечивают краткость записи запросов.

1. **Объединение (*UNION*)**. Объединение отношений R и S, обозначаемое как $R \cup S$, представляет собой множество кортежей, которые принадлежат R или S, либо им обоим – $R \cup S = \{t \mid t \in R \vee t \in S\}$.

2. **Разность (*MINUS*)**. Разностью отношений R и S, обозначаемой как $R–S$, называется множество кортежей, принадлежащих R, но не принадлежащих S $-R-S=\{t \mid t \in R \wedge t \notin S\}$.
3. **Декартово произведение (*TIMES*)**. Пусть R и S – отношения степени k_1 и k_2 соответственно. Тогда Декартовым произведением $R \times S$ отношений R и S называется множество всех кортежей степени k_1+k_2, первые k_1 компонентов которых образуют кортежи, принадлежащие R, а последние k_2 – кортежи, принадлежащие S, $-R \times S=\{r \parallel s \mid r \in R \wedge s \in S\}$. Здесь «||» – знак операции конкатенации кортежей.
4. **Проекция (*PROJECT*)**. Существо этой операции заключается в том, что берется отношение R, удаляются некоторые из его атрибутов и (или) переупорядочиваются оставшиеся атрибуты. Пусть R – отношение степени k. Обозначим через $\pi_{i_1,\ldots,i_m}(R)$, где i_j являются различными целыми в диапазоне от 1 до k, проекцию R на компоненты i_1, …, i_m, т.е. множество m-ок a_1, …, a_m, таких, что существует некоторый принадлежащий R кортеж b_1, …, b_k длины k, удовлетворяющий условию $a_j = b_{i_j}$ для j = 1, …, m: $\pi_{i_1,\ldots,i_m}(R)=\{< a_1,\ldots,a_m > \mid \exists < b_1,\ldots,b_k > (< b_1,\ldots,b_k > \in R \wedge \forall j = \overline{1,m}(a_j = b_{i_j}))\}$.

   Для строгого математического определения мы использовали порядковые номера атрибутов. Наряду с номерами в операциях проекции можно использовать имена атрибутов.
5. **Селекция (*SELECT*)**. Пусть F – формула, образованная: операндами, являющимися константами или номерами компонентов кортежей; операторами сравнения; логическими операторами. В этом случае результат операции селекции $\sigma_F(R)$ есть множество кортежей t, принадлежащих R, таких, что при подстановке i-го компонента t вместо всех вхождений номера i в формулу F для всех i она станет истинной. Наряду с номерами в операциях селекции можно использовать имена атрибутов.

Существует ряд полезных операций, которые хотя и могут быть выражены в терминах пяти ранее упоминавшихся (и поэтому являющихся дополнительными), но в литературе имеют специальные названия. Они используются иногда как примитивные операции.

1. **Пересечение (*INTERSECT*)**. Пересечение отношений R и S, обозначаемое как $R \cap S$, представляет собой множество кортежей, которые одновременно принадлежат и R, и S $-R \cap S=\{t \mid t \in R \wedge t \in S\}$. $R \cap S$ является сокращением выражения $R – (R – S)$.
2. **Частное (*DIVIDE*)**. Пусть R и S являются отношениями степени r и s соответственно, где $r > s$ и $S \neq \varnothing$. Тогда $R \div S$ есть множество кортежей t длины $(r - s)$, таких, что для всех кортежей u длины s, принадлежащих S, кортеж $t||u$ принадлежит R (знак «||» – знак операции конкатенации): $R \div S=\{t^{(r-s)} \mid \forall u^{(s)}(u \in S \rightarrow t \parallel u \in R)\}$. $R \div S$ является сокращением выражения $\pi_{1,\ldots,r-s}(R)-\pi_{1,\ldots,r-s}((\pi_{1,\ldots,r-s}(R) \times S)-R)$.
3. **Соединение (*JOIN*). Θ-соединение (*Θ-JOIN*)** R и S по атрибутам i и j, записываемое как $R \overset{\triangleright \triangleleft}{{}_{i\theta j}} S$, где Θ – оператор сравнения, есть краткая запись для $\sigma_{i\theta(r+j)}(R \times S)$, если R имеет степень r. Таким образом, Θ-соединение R и S представляет собой множество таких кортежей их Декартова произведения, что i-ый компонент R находится в отношении Θ с j-ым компонентом S. Если Θ является оператором «равно» («=»), эта операция

часто называется **эквисоединением (*EQUIJOIN*)**. Наряду с номерами в операциях соединения можно использовать имена атрибутов.

Операция соединения обобщается на случай, когда в соединении участвует не пара атрибутов i и j, а пара совместимых групп атрибутов X и Y – $R \overset{\triangleright\triangleleft}{_{X\theta Y}} S = \{r \,\|\, s \mid r \in R \wedge s \in S \wedge (\pi_X(r)\theta\pi_Y(s))\}$. Кортежи находятся в отношении Θ, тогда и только тогда, когда в этом отношении находятся все пары их соответствующих компонентов.

Естественным соединением (*NATURAL JOIN* или *INNER JOIN*) называется эквисоединение, из результата которого исключены по одному экземпляру из каждой пары совпадающих атрибутов (атрибуты группы X или группы Y).

Композиционным соединением (*COMPOSITE JOIN*) называется эквисоединение, при котором атрибуты соединения (атрибуты групп X и Y) не включаются в результат. Такое соединение уместно при использовании для связей кортежей суррогатных ключей.

Левым внешним соединением (*LEFT OUTER JOIN*) называется соединение, при котором к результату эквисоединения добавляются кортежи левого операнда, не вошедшие в эквисоединение, конкатенированные справа с заполненными неопределенными значениями (*NULL*) кортежами степени, равной степени правого операнда.

Правым внешним соединением (*RIGHT OUTER JOIN*) называется соединение, при котором к результату эквисоединения добавляются кортежи правого операнда, не вошедшие в эквисоединение, конкатенированные слева с заполненными неопределенными значениями (*NULL*) кортежами степени, равной степени левого операнда.

Результат **полного внешнего соединения (*FULL OUTER JOIN*)** совпадает с объединением левого и правого внешних соединений.

Примеры

R

A	B
1	2
1	4
3	4

S

A	B
2	1
5	6
3	4

$R \cup S$

A	B
1	2
1	4
2	1
3	4
5	6

$R - S$

A	B
1	2
1	4

$R \cap S$

A	B
3	4

$\pi_2(R)$

B
2
4

$\sigma_{A=1}(R)$

A	B
1	2
1	4

На слайде показаны примеры совместимых отношений *R* и *S*, а также результаты теоретико-множественных операций и операций проекции и селекции. Обратите внимание на нормализованные результаты операций объединения и проекции. В них отсутствуют кортежи-дубликаты.

Примеры

R

A	B	C	D
1	2	5	6
2	3	7	8
3	4	5	6

S

X	Y	Z
5	6	6
7	8	8

$R \times S$

A	B	C	D	X	Y	Z
1	2	5	6	5	6	6
1	2	5	6	7	8	8
2	3	7	8	5	6	6
2	3	7	8	7	8	8
3	4	5	6	5	6	6
3	4	5	6	7	8	8

$R \overset{\triangleright\triangleleft}{}_{\{C,D\}=\{X,Y\}} S$

A	B	C	D	X	Y	Z
1	2	5	6	5	6	6
2	3	7	8	7	8	8
3	4	5	6	5	6	6

На слайде показаны примеры отношений *R* и *S*, а также результаты операций Декартова произведения и эквисоединения по совместимым группам атрибутов.

Примеры

R

A	B	C	D
1	2	5	6
1	2	7	8
2	3	5	6
2	3	7	8
3	4	5	6

S

C	D
5	6
7	8

$R \div S$

A	B
1	2
2	3

На слайде показаны примеры отношений *R* и *S*, а также результат операции взятия частного.

Для записи запросов на языке реляционной алгебры мы будем использовать не введенные ранее обозначения операций, а их словесные аналоги, предназначенные для задания запросов в компьютеризированных языках манипулирования данными.

Общий линейный синтаксис операций реляционной алгебры в таком языке имеет вид <*операция*> → <*имя результирующего отношения*>. Приведем синтаксис операций в этом языке.

Объединение.

UNION <имя первого отношения> AND <имя второго отношения>.

Операнды теоретико-множественных операций *UNION*, *MINUS* и *INTERSECT* должны иметь совпадающие с точностью до имен атрибутов заголовки.

Разность.

MINUS <имя «уменьшаемого» отношения> AND <имя «вычитаемого» отношения>.

Декартово произведение.

TIMES <имя первого отношения> AND <имя второго отношения>.

Операнды операции *TIMES* не должны иметь атрибутов с совпадающими именами.

Проекция.

PROJECT <имя отношения> OVER <список через запятую имен атрибутов, на которые осуществляется проекция>.

Селекция.

SELECT <имя отношения> WHERE <логическое условие>.

<*логическое условие*> составляют:

- имена атрибутов указанного отношения,
- константы,
- операции над значениями (арифметические, строковые),
- операции сравнения (=, >, …),
- логические операции (*NOT, AND, OR*)

Пересечение.

INTERSECT <имя первого отношения> AND <имя второго отношения>.

Частное.

DIVIDE <имя делимого отношения> BY <имя отношения-делителя> OVER <список имен «общих» атрибутов>.

У операции деления в этом языке есть следующие ограничения:

- «общие» атрибуты должны иметь совпадающие имена в отношениях-операндах;
- в списке «общих» атрибутов имена не дублируются;
- в отношении-делителе кроме «общих» атрибутов других атрибутов нет;
- множество атрибутов отношения-результата представляет собой разность «множество атрибутов делимого отношения» - «множество атрибутов отношения-делителя».

Естественное соединение.

JOIN <имя первого отношения> AND <имя второго отношения> OVER <список имен «общих» атрибутов>.

У операции естественного соединения в этом языке есть следующие ограничения:

- «общие» атрибуты должны иметь совпадающие имена в отношениях-операндах;
- в списке «общих» атрибутов имена не дублируются;
- «необщие» атрибуты отношений-операндов не имеют совпадений в именах.

Реляционная алгебра. Примеры запросов

Получить фамилии хирургов

```
SELECT ВРАЧ WHERE Специальность = 'ХИРУРГ' -> TEMP
PROJECT TEMP OVER Фамилия -> RESULT
```

Получить фамилии врачей, лечащих больных палаты №2 больницы №5

```
SELECT РАЗМЕЩЕНИЕ WHERE К/Б = 5 AND Н/П = 2 -> T1
PROJECT T1 OVER Р/Н -> T2
JOIN T2 AND ВРАЧ-ПАЦИЕНТ OVER Р/Н -> T3
PROJECT T3 OVER К/В -> T4
JOIN T4 AND ВРАЧ OVER К/В -> T5
PROJECT T5 OVER Фамилия -> RESULT
```

Получить фамилии врачей, лечащих всех больных палаты №2 больницы №5

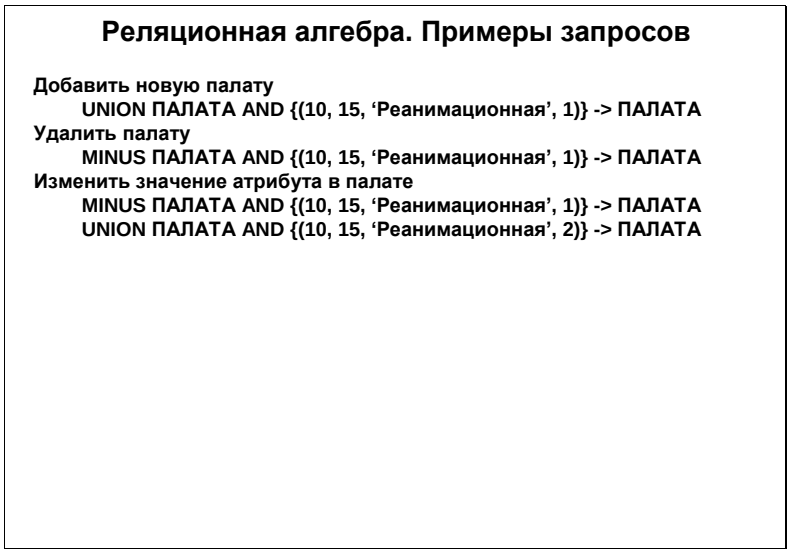
```
SELECT РАЗМЕЩЕНИЕ WHERE К/Б = 5 AND Н/П = 2 -> T1
PROJECT T1 OVER Р/Н -> T2
DIVIDE ВРАЧ-ПАЦИЕНТ BY T2 OVER Р/Н -> T3
JOIN T3 AND ВРАЧ OVER К/В -> T4
PROJECT T4 OVER Фамилия -> RESULT
```

На слайде приведены примеры реализации запросов выборки информации на языке реляционной алгебры. Каждый запрос представляет собой последовательность операций над базовыми и производными отношениями. Результатом последней операции является целевое отношение запроса – *RESULT*.

Многочисленные исследования языков реляционной алгебры породили разнообразные идеи оптимизации запросов. Многие из них очевидны. Например, перед выполнением дорогостоящих операций (*TIMES, JOIN, DIVIDE*) необходимо с помощью унарных операций (*PROJECT, SELECT*) избавиться от ненужной информации (атрибутов, кортежей). Хорошим стилем формулирования запросов считается тот, при котором учитывается это обстоятельство.

Обратите внимание на то, что прежде чем делать соединение кортежей отношений *РАЗМЕЩЕНИЕ* и *ВРАЧ-ПАЦИЕНТ* во втором запросе, мы с помощью операций селекции и проекции получили регистрационные номера пациентов, лежащих в указанной палате. Для нашего запроса эта информация является минимально необходимой для последующих действий. Надеюсь, всем очевидно, что соединение отношения *T2* с отношением *ВРАЧ-ПАЦИЕНТ* потребует гораздо меньше памяти и времени, нежели соединение отношений *РАЗМЕЩЕНИЕ* и *ВРАЧ-ПАЦИЕНТ*.

Следующее обстоятельство также позволит сформулировать оптимальный запрос. Тщательно выбирайте базовые отношения, достаточные для получения необходимого результата. Не используйте в запросе отношения, информация которых является избыточной. Так, во втором запросе кому-то могло показаться, что первой необходимо использовать селекцию кортежа в отношении *ПАЛАТА*. Это, конечно, неправильно. На слайде представлен эквивалентный, но, несомненно, более оптимальный вариант. Аналогичное можно сказать и по поводу избыточного соединения отношения *T2* с отношением *ПАЦИЕНТ* и последующего соединения результата с отношением *ВРАЧ-ПАЦИЕНТ*.



Язык реляционной алгебры обычно не используют для изменения состояния БД, однако действия включения, модификации и удаления кортежей можно выполнить и в этом языке. На слайде приведены примеры таких действий. Особенностями запросов, изменяющих экстенсионал базового отношения, являются:

- действие включения выполняется операцией *UNION*;
- действие удаления выполняется операцией *MINUS*;
- действие модификации сводится к действию удаления и последующего включения;
- во всех этих случаях первым операндом и результатом является изменяемое базовое отношение;
- вторым операндом этих операций является либо производное отношение, полученное запросом выборки, либо отношение, кортежи которого заданы литерально (явным указанием их компонентов).

4.2.4.2. Реляционное исчисление с переменными-кортежами

Реляционное исчисление с переменными-кортежами

Запрос – $\{t \mid \psi(t)\}$

Атомы:

1. $R(s)$
2. $s[i]\Theta u[j]$
3. $s[i]\Theta a$ **и** $a\Theta s[i]$

Правила образования формул:

1. **Атом – формула.**
2. **Если** ψ_1 **и** ψ_2 **- формулы, то** $\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2, \neg\psi_1$ **- тоже формулы.**
3. **Если** ψ **- формула, то** $\exists s(\psi)$ **- тоже формула.**
4. **Если** ψ **- формула, то** $\forall s(\psi)$ **- тоже формула.**
5. **Приоритет операций –** $\Theta, \begin{Bmatrix}\exists\\ \forall\end{Bmatrix}, \neg, \wedge, \vee$

   Круглые скобки для изменения приоритета.
6. **Ничто иное не является формулой.**

Основу реляционных исчислений составляет исчисление предикатов первого порядка. В зависимости от того, какие области значений имеют переменные исчисления, выделяют две разновидности реляционных исчислений: реляционное исчисление с переменными-кортежами и реляционное исчисление с переменными на доменах. В первом случае переменные означивают кортежи некоторого Декартова произведения доменов, во втором – значения одного домена. Определим первое исчисление.

Запросы в реляционном исчислении с переменными-кортежами имеют вид $\{t \mid \psi(t)\}$, где t – переменная-кортеж, т.е. переменная, обозначающая кортеж некоторой фиксированной длины, а ψ – формула, построенная из атомов и совокупности операторов, которые определяются ниже. Отметим, что есть важное дополнительное условие того, чтобы указанная конструкция являлась запросом реляционного исчисления, но об этом мы поговорим позже.

Атомы формул ψ могут быть трех типов.

1. $R(s)$, где R – имя отношения, а s – переменная-кортеж. Этот атом принимает значение «*истина*», когда s есть кортеж отношения R.
2. $s[i]\ \Theta\ u[j]$, где s и u являются переменными-кортежами, а Θ – оператор сравнения. Этот атом принимает значение «*истина*», когда i-ый компонент s находится в отношении Θ с j-ым компонентом u.
3. $s[i]\ \Theta\ a$ или $a\ \Theta\ s[i]$, где Θ и $s[i]$ имеют тот же смысл, а a – это константа. Этот атом принимает значение «*истина*», когда i-ый компонент s находится в отношении Θ с константой a.

При определении формул реляционного исчисления необходимо ввести понятия «свободных» и «связанных» переменных-кортежей. Эти понятия имеют в точности тот же смысл, что и в исчислении предикатов. Неформально вхождение переменной в формулу является «связанным», если этой переменной предшествует квантор «для всех» или «существует». В противном случае мы называем переменную «свободной». Для однозначности будем использовать круглые скобки для указания областей связности переменных.

Формулы, а также свободные и связанные вхождения переменных-кортежей в эти формулы определяются рекурсивно следующим образом.

1. Каждый атом есть формула. Все вхождения переменных-кортежей, упомянутые в атоме, являются свободными в этой формуле.

2. Если ψ_1 и ψ_2 – формулы, то $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$ и $\neg\psi_1$ – тоже формулы, утверждающие соответственно, что «ψ_1 и ψ_2 обе являются истинными», «ψ_1 или ψ_2, либо обе истинны» и «ψ_1 не истинна». Экземпляры переменных-кортежей не меняют своего статуса «связана-свободна» в новых формулах. Заметим, что некоторое вхождение переменной s может быть связанным в ψ_1, в то время как другое – свободным в ψ_2. В новой формуле, например, $\psi_1 \wedge \psi_2$, эта переменная будет иметь статус «свободна».
3. Если ψ – формула, в которой есть свободная переменная s, то $\exists s\ (\psi)$ – также формула. Символ «$\exists$» представляет собой квантор существования. Вхождения переменной s, свободные в формуле ψ, становятся связанными этим квантором в новой формуле. Остальные вхождения переменных-кортежей, включая возможные вхождения s, связанные в ψ, своего статуса не меняют. Формула $\exists s\ (\psi)$ утверждает, что существует значение s, при подстановке которого вместо всех свободных вхождений s в формулу ψ эта формула становится истинной. Например, $\exists s\ (R(s))$ означает, что отношение R не пусто.
4. Если ψ – формула, в которой есть свободная переменная s, то $\forall s\ (\psi)$ – также формула. Символ «$\forall$» представляет собой квантор всеобщности. Вхождения переменной s, свободные в формуле ψ, становятся связанными этим квантором в новой формуле. Остальные вхождения переменных-кортежей, включая возможные вхождения s, связанные в ψ, своего статуса не меняют. Формула $\forall s\ (\psi)$ утверждает, что какое бы значение подходящей степени мы не подставили вместо всех свободных вхождений s в формулу ψ, эта формула становится истинной.
5. Формулы могут заключаться в круглые скобки для изменения приоритета операторов. Предполагается следующий порядок старшинства операторов – $\Theta, \begin{Bmatrix}\exists \\ \forall\end{Bmatrix}, \neg, \wedge, \vee$.
6. Ничто иное не является формулой.

Запрос реляционного исчисления с переменными-кортежами есть выражение вида $\{t \mid \psi(t)\}$, где t – единственная свободная переменная-кортеж в формуле ψ.

Отметим, что хотя в определении абстрактного реляционного исчисления использовались только основные логические операции конъюнкции, дизъюнкции и отрицания, в реальных языках часто к ним добавляются операции импликации и эквивалентности, позволяющие более кратко и, главное, доступнее для человека выразить логические высказывания.

Реляционное исчисление с переменными-кортежами

Реализация основных операций реляционной алгебры

Объединение – $R \cup S = \{t \mid R(t) \vee S(t)\}$

Разность – $R - S = \{t \mid R(t) \wedge \neg S(t)\}$

Декартово произведение –

$$R^{(r)} \times S^{(s)} = \{t^{(r+s)} \mid \exists u^{(r)} \exists v^{(s)} (R(u) \wedge S(v) \wedge t[1] = u[1] \wedge \ldots$$
$$\ldots \wedge t[r] = u[r] \wedge t[r+1] = v[1] \wedge \ldots \wedge t[r+s] = v[s])\}$$

Проекция –

$$\pi_{i_1, i_2, \ldots i_k}(R) = \{t^{(k)} \mid \exists u (R(u) \wedge t[1] = u[i_1] \wedge \ldots \wedge t[k] = u[i_k])\}$$

Селекция – $\sigma_F(R) = \{t \mid R(t) \wedge F'\}$,

где *F'*- это *F*, в которой *i* заменено на *t[i]*

Мы уже упоминали о том, что реляционные спецификационные языки, введенные Коддом (реляционная алгебра и оба вида реляционных исчислений), обладают одинаковой мощностью выражения запросов. На слайде показаны выражения реляционного исчисления с переменными-кортежами (правые части формул), результат которых совпадает с результатами основных операций реляционной алгебры (левые части формул).

Реляционное исчисление с переменными-кортежами. Язык ALPHA. Примеры запросов

Получить полные сведения о больницах

$\{t \mid БОЛЬНИЦА(t)\}$

GET W (БОЛЬНИЦА)
GET W (БОЛЬНИЦА.К/Б, БОЛЬНИЦА.Название, БОЛЬНИЦА.Адрес, БОЛЬНИЦА.Ч/К)

Получить фамилии хирургов

$\{t \mid ВРАЧ(t) \wedge t[\text{Специальность}] = \text{'ХИРУРГ'}\}$

GET W (ВРАЧ.Фамилия):
ВРАЧ.Специальность = 'ХИРУРГ'

Коддом был предложен язык **ALPHA**, относящийся к классу языков реляционного исчисления с переменными-кортежами. Мы рассмотрим диалект языка ALPHA, опубликованный в монографии Дейта.

Реализации двух первых запросов, показанные на слайде, приведены как для абстрактного языка исчисления, рассмотренного ранее, так и для языка ALPHA. На этих примерах хорошо видны основные отличия в запросах на этих языках.

Если в первом языке явно вводятся свободные переменные запроса и атомы первого вида, определяющие фактически некоторые отношения, как области значений этих переменных, то во втором языке они опущены. Действительно, конструкция *GET W (БОЛЬНИЦА)* в первом запросе неявно вводит одну свободную переменную *t* и атом *БОЛЬНИЦА*(*t*). В общем случае (это будет показано в примерах) в запросе на языке ALPHA можно использовать несколько переменных-кортежей, определенных на одних и тех же или различных отношениях. Для каждой из них неявно предполагается соответствующий атом.

В логическом выражении запроса на языке ALPHA ссылка на неявную свободную переменную осуществляется с помощью имени отношения, являющегося ее областью значений. Это можно увидеть во втором запросе. Кстати, для указания компонента переменной-кортежа используется синтаксис *<имя переменной>.<имя атрибута>*. Порядковые номера атрибутов в языке ALPHA не используются.

Второй вариант первого запроса явно задает список атрибутов, чьи значения будут представлены в результате (на эти атрибуты будет выполнена окончательная проекция результирующей таблицы). В данном случае этот список совпадает с набором атрибутов отношения *БОЛЬНИЦА*, и поэтому возможна краткая форма первого варианта.

Сначала мы рассмотрим осуществление действия выборки (операция *GET*), а затем поговорим о том, как выполняются другие действия в этом языке.

В рабочую область с именем *W* попадают данные, отобранные в результате выполнения операции. То есть *W* содержит отношение, выделенное из БД посредством предложения *GET*. После завершения операции *GET* к содержимому рабочей области *W* можно обращаться как к отношению с именем *W*.

Рассмотрим еще несколько запросов, на примере которых покажем дополнительные грамматические конструкции языка ALPHA.

Получить названия и число коек палат больницы с кодом 5, упорядочить их по возрастанию числа коек

GET W (ПАЛАТА.Название, ПАЛАТА.Ч/К):

ПАЛАТА.К/Б = 5 UP ПАЛАТА.Ч/К

Запрос демонстрирует возможность сортировки результата по возрастанию (*UP*) или убыванию (*DOWN*) значений атрибутов. Необязательно, чтобы в списках сортировки, указываемых после логического выражения (если оно есть), использовались целевые атрибуты.

Получить регистрационный номер и фамилию самого молодого пациента
GET W (1) (ПАЦИЕНТ.Р/Н, ПАЦИЕНТ.Фамилия):
DOWN ПАЦИЕНТ.Д/Р

В этом запросе помимо сортировки (кстати, логическое выражение в нем не понадобилось) используется квотирование результата (конструкция «*(1)*» после имени рабочей области). Можно указать любое интересующее вас количество строк результата, большее 0. Если мощность результирующей таблицы меньше указанной квоты, выдаются все строки таблицы.

Получить коды врачей, лечащих пациента с регистрационным номером 111111
GET W (ВРАЧ-ПАЦИЕНТ.К/В):
ВРАЧ-ПАЦИЕНТ.Р/Н = 111111

RANGE ВРАЧ-ПАЦИЕНТ X
GET W (X.К/В):
X.Р/Н = 111111

Можно в явном виде ввести переменную-кортеж запроса с помощью конструкции *RANGE*, имеющей следующий синтаксис: *RANGE <имя отношения> <имя переменной>*. Причем вводимая таким образом переменная может быть как свободной, так и связанной в запросе. В последнем случае она всегда должна быть введена явно.

Последний пример демонстрирует два эквивалентных варианта одного и того же запроса. Его единственная свободная переменная-кортеж в первом случае введена неявно, а во втором случае – явно.

Получить фамилии врачей, лечащих пациента с регистрационным номером 111111
RANGE ВРАЧ-ПАЦИЕНТ X
GET W (ВРАЧ.Фамилия):

$\exists X\ (X.К/В = ВРАЧ.К/В \wedge X.Р/Н = 111111)$

В этом запросе нам уже не обойтись без явного определения переменной запроса. Кстати, здесь же видно, что логическое выражение запроса в языке ALPHA использует традиционный синтаксис, включающий операции сравнения, логические операции и кванторы.

Получить фамилии врачей, лечащих женщин (вариант 1)
RANGE ПАЦИЕНТ X
GET W1 (ВРАЧ-ПАЦИЕНТ.К/В):

$\exists X\ (X.Р/Н = ВРАЧ\text{-}ПАЦИЕНТ.Р/Н \wedge X.Пол = \text{‘ЖЕНСКИЙ’})$

RANGE W1 WX
GET W2 (ВРАЧ.Фамилия):

$\exists WX\ (WX.К/В = ВРАЧ.К/В)$

В этом и последующем примерах показаны две реализации одного и того же запроса. В первом варианте весь запрос разбит на два подзапроса, причем второй использует результирующее отношение первого.

Получить фамилии врачей, лечащих женщин (вариант 2)
RANGE ПАЦИЕНТ X
RANGE ВРАЧ-ПАЦИЕНТ Y
GET W (ВРАЧ.Фамилия):

$$\exists Y\ (Y.К/В = ВРАЧ.К/В \wedge \exists X\ (X.Р/Н = Y.Р/Н \wedge X.Пол = \text{‘ЖЕНСКИЙ’}))$$

или

$$\exists X\ \exists Y\ (Y.К/В = ВРАЧ.К/В \wedge X.Р/Н = Y.Р/Н \wedge X.Пол = \text{‘ЖЕНСКИЙ’})$$

Второй вариант получает результат одной командой *GET*, использующей одну неявную свободную переменную и две явно введенных связанных переменных. В примере показаны два эквивалентных вида логических выражений. Первое отражает «ход мысли» человека. Действительно, оно является переводом на логический язык следующей фразы: «Выдать фамилии врачей, для которых существует связь с пациентом, являющимся женщиной». Второй вид «более нормален» с точки зрения логики – он удовлетворяет требованиям предваренной нормальной формы:

- все кванторные выражения собраны в начале выражения в единый префикс;
- остальная часть выражения (матрица) кванторных выражений не содержит.

Более того, матрица выражения находится в конъюнктивной нормальной форме.

Получить фамилии врачей, которые лечат тех же пациентов, что и врач с кодом 999
RANGE ВРАЧ-ПАЦИЕНТ X
RANGE ВРАЧ-ПАЦИЕНТ Y
GET W (ВРАЧ.Фамилия):

$$\exists X\ (X.К/В = ВРАЧ.К/В \wedge \exists Y\ (Y.Р/Н = X.Р/Н \wedge Y.К/В = 999))$$

Получить список фамилий больных с их диагнозами
GET W (ПАЦИЕНТ.Фамилия, ДИАГНОЗ.Т/Д):
ДИАГНОЗ.Р/Н = ПАЦИЕНТ.Р/Н

Пример запроса с двумя неявными свободными переменными-кортежами: первая пробегает по кортежам отношения *ПАЦИЕНТ*, вторая – *ДИАГНОЗ*. Логическое выражение запроса фактически задает естественное соединение отношений, а целевой список – проекцию полученного в результате соединения отношения.

Если в данном случае опустить логическое выражение, вместо соединения будет выполнена операция Декартова произведения отношений.

Получить фамилии врачей, которые не лечат пациента с регистрационным номером 111111
RANGE ВРАЧ-ПАЦИЕНТ X
GET W (ВРАЧ.Фамилия):

$$\forall X\ (X.К/В \neq ВРАЧ.К/В \vee X.Р/Н \neq 111111)$$

или

$$\neg\ \exists X\ (X.К/В = ВРАЧ.К/В \wedge X.Р/Н = 111111)$$

Предыдущие запросы, если и использовали кванторы, то только кванторы существования. В этом запросе впервые понадобился квантор всеобщности. В первом варианте логического выражения он представлен явно, во втором, эквивалентном первому, варианте – неявно.

Второй вариант ближе «человеческой логике», поскольку дословно транслирует на язык логики следующую фразу: «Выдать фамилии врачей, у которых не существует связей с пациентом с регистрационным номером, равным 111111». Смысл первого выражения можно выразить высказыванием «Выдать фамилии врачей таких, что для любой связи *ВРАЧ-ПАЦИЕНТ* либо она относится к другому врачу, либо в ней участвует

пациент, регистрационный номер которого не равен 111111», на наш взгляд менее очевидным для человека.

Получить фамилии пациентов, которые лечатся у всех врачей
RANGE ВРАЧ-ПАЦИЕНТ X
RANGE ВРАЧ Y
GET W (ПАЦИЕНТ.Фамилия):

$$\forall Y\, \exists X\, (X.Р/Н = ПАЦИЕНТ.Р/Н \wedge X.К/В = Y.К/В)$$

Получить регистрационные номера пациентов, которые лечатся у всех врачей
RANGE ВРАЧ-ПАЦИЕНТ X
RANGE ВРАЧ Y
GET W (ПАЦИЕНТ.Р/Н):

$$\forall Y\, \exists X\, (X.Р/Н = ПАЦИЕНТ.Р/Н \wedge X.К/В = Y.К/В)$$

неверно:

GET W (X.Р/Н): $\forall Y\, \exists X\, (X.К/В = Y.К/В)$

GET W (X.Р/Н): $\forall Y\, (X.К/В = Y.К/В)$

Небольшое видоизменение предыдущего запроса может повлечь за собой желание «оптимизировать» логическое выражение. В примере сначала показан синтаксически и семантически правильный запрос, а затем приведены два неверных запроса.

В первом из них допущена грубая синтаксическая ошибка – в нем отсутствуют свободные переменные. Второй из неправильных запросов удовлетворяет синтаксису языка, но не верен по смыслу. Единственный вариант состояния БД, при котором этот запрос может выдать что-то в результате, предполагает наличие единственного кортежа в отношении *ВРАЧ*. Что для реальной медицинской БД вряд ли возможно.

Получить фамилии пациентов, которые лечатся у всех практикующих врачей
RANGE ВРАЧ-ПАЦИЕНТ X
RANGE ВРАЧ-ПАЦИЕНТ Y
GET W (ПАЦИЕНТ.Фамилия):

$$\forall Y\, \exists X\, (X.Р/Н = ПАЦИЕНТ.Р/Н \wedge X.К/В = Y.К/В)$$

Еще одно видоизменение запроса приводит к смене области значений переменной *Y*, ею стало отношение *ВРАЧ-ПАЦИЕНТ*, поскольку именно в нем представлены коды практикующих врачей.

Получить коды врачей, лечащих по крайней мере тех же пациентов, что и врач с кодом 999
RANGE ПАЦИЕНТ X
RANGE ВРАЧ-ПАЦИЕНТ Y
RANGE ВРАЧ-ПАЦИЕНТ Z
GET W (ВРАЧ.К/В):

$$\forall X\, (\exists Y\, (Y.К/В = 999 \wedge Y.Р/Н = X.Р/Н) \rightarrow$$

$$\exists Z\, (Z.К/В = ВРАЧ.К/В \wedge Z.Р/Н = X.Р/Н))$$

Последний пример запроса на языке ALPHA, осуществляющего действие выборки, демонстрирует использование логической операции импликации. Наряду с операцией эквивалентности, эта операция позволяет придать некоторым логическим выражениям «более человеческую» форму. Действительно, приведенное логическое выражение может быть на словах представлено следующим образом: «Выдать коды таких врачей, что для любого пациента справедливо следующее – если он ходил к 999-му врачу, то он ходил и к

этому врачу». Что собственно и требовалось в запросе. Поверьте, что логическое выражение, в котором произведено тождественное преобразование с заменой операции импликации на операции отрицания и дизъюнкции, будет выглядеть менее естественным для людей.

Для того, чтобы изменить кортежи в отношении, пользователь должен выбрать их, используя предложение *HOLD*, модифицировать их в рабочей области *W* и затем поместить обратно в БД с использованием предложения *UPDATE W*. Действие предложения *HOLD* в основном аналогично действию предложения *GET*, но это предложение сообщает СУБД о том, что пользователь намеревается изменить соответствующие кортежи. Это, например, может предотвратить доступ другого, одновременно работающего пользователя к этим кортежам для выполнения операции, отличной от выборки, до тех пор, пока не будет завершена операция обновления.

Если после использования предложения *HOLD* пользователь принимает решение не обновлять данные, он может завершить операцию путем использования предложения *RELEASE W*.

Кстати, следует отметить, что целевой список в предложении *HOLD* должен содержать первичный ключ отношения. Несмотря на это требование, последовательность предложений *HOLD-UPDATE* нельзя использовать для изменения значения первичного ключа в кортеже. Если такое изменение необходимо, пользователь должен удалить старый кортеж с использованием предложения *DELETE* и затем поместить в отношение новый кортеж при помощи предложения *PUT*.

Изменить значение атрибута *Число коек* у больницы с кодом 2, сделать его равным 100

```
HOLD W (БОЛЬНИЦА.К/Б, БОЛЬНИЦА.Ч/К):
        БОЛЬНИЦА.К/Б = 2
W.Ч/К = 100
UPDATE W
```

Изменить код больницы с 2 на 20

```
HOLD W (БОЛЬНИЦА):
        БОЛЬНИЦА.К/Б = 2
DELETE W
W.К/Б = 20
PUT W (БОЛЬНИЦА)
```

Для включения нового кортежа в отношение его надо сначала сформировать в рабочей области, а затем, используя то же предложение *PUT*, сохранить его в БД.

Добавить в БД новую больницу

```
W.К/Б = 10
W.Название = 'Госпиталь Святой Елены'
W.Адрес = 'Торонто'
W.Ч/К = 200
PUT W (БОЛЬНИЦА)
```

Кортежи, помещенные в рабочую область предложением *HOLD*, могут быть удалены из БД предложением *DELETE*.

Удалить больницу с кодом 5

```
HOLD W (БОЛЬНИЦА):
        БОЛЬНИЦА.К/Б = 5
```

DELETE W

Удалить все анализы
HOLD W (АНАЛИЗ)
DELETE W

В заключение рассмотрим, какие преимущества имеет язык ALPHA по сравнению с навигационными языками.

Мощность. Этот язык (наряду с языком реляционной алгебры) является эталоном так называемой реляционной полноты. Реляционная полнота есть основная мера селективной мощности спецификационного реляционного языка данных. Говорят, что спецификационный язык обладает реляционной полнотой, если любое отношение, выводимое из БД посредством выражения реляционного исчисления, может быть получено с использованием этого языка.

Экономность, с которой эта полнота достигается (любое выводимое отношение может быть получено с помощью одного предложения языка), является фактором, который делает язык ALPHA столь мощным.

Простота. Исчисление, как и алгебра, предоставляет довольно простую форму задания запросов, хотя простота рассматриваемого языка может быть не сразу очевидной для читателя, который столкнулся с ним впервые, особенно если он не знаком с формальной нотацией исчисления предикатов первого порядка. Важно отметить, что в общем случае в языке ALPHA сложность предложения прямо пропорциональна сложности операции, которую пользователь пытается выполнить. В частности, простые операции могут выражаться весьма просто. Какой из языков является более простым – в значительной мере дело вкуса. Программисты часто отдают предпочтение алгебре, непрограммисты, владеющие математической логикой, – исчислению.

Непроцедурность. Предложения языка ALPHA представляют собой простые описания того, что нужно получить. Например, предложение *GET* является описанием требуемых данных; оно не определяет, как эти данные должны быть найдены.

4.2.4.3. Реляционное исчисление с переменными на доменах

Реляционное исчисление с переменными на доменах

Запрос – $\{x_1x_2...x_k \mid \psi(x_1, x_2,..., x_k)\}$

Атомы:

1. $R(x_1x_2...x_k)$
2. $x\Theta y$

Правила образования формул:

1. **Атом – формула.**
2. **Если** ψ_1 **и** ψ_2 **- формулы, то** $\psi_1 \wedge \psi_2, \psi_1 \vee \psi_2, \neg\psi_1$ **- тоже формулы.**
3. **Если** ψ **- формула, то** $\exists x(\psi)$ **- тоже формула.**
4. **Если** ψ **- формула, то** $\forall x(\psi)$ **- тоже формула.**
5. **Приоритет операций –** $\Theta, \begin{Bmatrix}\exists\\ \forall\end{Bmatrix}, \neg, \wedge, \vee$

   Круглые скобки для изменения приоритета.
6. **Ничто иное не является формулой.**

Запросы в реляционном исчислении с переменными на доменах имеют вид $\{x_1x_2...x_k \mid \psi(x_1, x_2,..., x_k)\}$, где x_1, x_2, …, x_k – переменные на доменах, т.е. переменные, обозначающие скалярные значения, взятые из определенных доменов, а ψ – формула, построенная из атомов и совокупности операторов, которые определяются ниже. Позже мы введем важное дополнительное условие того, чтобы указанная конструкция являлась запросом реляционного исчисления с переменными на доменах.

Атомы формул ψ могут быть двух типов.

1. $R(x_1\ x_2\ \ldots\ x_k)$, где R – имя отношения степени k, а каждое x_i есть константа или переменная на домене. Этот атом принимает значение «*истина*», когда $x_1\ x_2\ \ldots\ x_k$ есть кортеж отношения R.
2. $x\ \Theta\ y$, где x и y являются константами или переменными на доменах, а Θ – оператор сравнения. Этот атом принимает значение «*истина*», когда x находится в отношении Θ с y.

Формулы, а также свободные и связанные вхождения переменных на доменах в эти формулы определяются рекурсивно следующим образом.

1. Каждый атом есть формула. Все вхождения переменных на доменах, упомянутые в атоме, являются свободными в этой формуле.
2. Если ψ_1 и ψ_2 – формулы, то $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$ и $\neg\psi_1$ – тоже формулы, утверждающие соответственно, что «ψ_1 и ψ_2 обе являются истинными», «ψ_1 или ψ_2, либо обе истинны» и «ψ_1 не истинна». Экземпляры переменных на доменах не меняют своего статуса «связана-свободна» в новых формулах. Заметим, что некоторое вхождение переменной x может быть связанным в ψ_1, в то время как другое – свободным в ψ_2. В новой формуле, например, $\psi_1 \wedge \psi_2$, эта переменная будет иметь статус «свободна».
3. Если ψ – формула, в которой есть свободная переменная x, то $\exists x\ (\psi)$ – также формула. Символ «$\exists$» представляет собой квантор существования. Вхождения переменной x, свободные в формуле ψ, становятся связанными этим квантором в новой формуле. Остальные вхождения переменных на доменах, включая возможные вхождения x, связанные в ψ, своего статуса не меняют. Формула $\exists x\ (\psi)$ утверждает, что существует значение x, при

подстановке которого вместо всех свободных вхождений x в формулу ψ эта формула становится истинной.

4. Если ψ – формула, в которой есть свободная переменная x, то $\forall x\,(\psi)$ – также формула. Символ «$\forall$» представляет собой квантор всеобщности. Вхождения переменной x, свободные в формуле ψ, становятся связанными этим квантором в новой формуле. Остальные вхождения переменных на доменах, включая возможные вхождения x, связанные в ψ, своего статуса не меняют. Формула $\forall x\,(\psi)$ утверждает, что какое бы значение мы не подставили вместо всех свободных вхождений x в формулу ψ, эта формула становится истинной.
5. Формулы могут заключаться в круглые скобки для изменения приоритета операторов. Предполагается следующий порядок старшинства операторов – $\Theta, \begin{Bmatrix}\exists\\ \forall\end{Bmatrix}, \neg, \wedge, \vee$.
6. Ничто иное не является формулой.

Запрос реляционного исчисления с переменными на доменах есть выражение вида $\{x_1 x_2 ... x_k \mid \psi(x_1, x_2, ..., x_k)\}$, где x_1, x_2, …, x_k – свободные переменные на доменах в формуле ψ, и других свободных переменных в этой формуле нет.

В качестве примера языка реляционного исчисления с переменными на доменах рассмотрим замечательный во многих смыслах **язык запросов на примерах (Query-By-Example – QBE)**. Этот язык был разработан М. М. Злуфом (M. M. Zloof) в научно-исследовательской лаборатории фирмы IBM. Его описание впервые было опубликовано в журнале «IBM Systems Journal» в 1977 году.

Перечислим основные отличительные особенности языка QBE.

- Ориентация на диалоговое взаимодействие человека с системой БД.

Он предназначен для интерактивного взаимодействия человека с системой БД посредством дисплея, мыши и клавиатуры. В связи с этим открываются дополнительные возможности в виде совместных действий человека и системы по формулированию запросов на этом языке.

- Двумерный синтаксис команд.

Всякая операция в QBE задается с помощью одной или нескольких таблиц-шаблонов. Поскольку операции задаются в табличной форме, говорят, что QBE имеет двумерный синтаксис. Все остальные реляционные языки имеют линейный синтаксис. Значительной особенностью QBE является использование примеров для спецификации запросов. Основная идея состоит в том, что пользователь формирует запрос, занося пример возможного ответа в соответствующее место пустой таблицы-шаблона.

- Минимум вводимых человеком символов.

Как уже отмечалось, каждая таблица запроса строится на экране дисплея частично человеком, частично системой. Таким образом, многие элементы запроса (имена таблиц, имена атрибутов) подставляет система, пользователю остается заполнить лишь минимально необходимые поля тела таблицы-шаблона. Но и эта задача сводится к вводу только действительно необходимой информации – констант, операций сравнения и действий. Благодаря двумерному синтаксису в QBE нет необходимости в ключевых словах, словах-разделителях, прочих специальных терминальных символах, ввод которых требует дополнительных усилий пользователя при линейном синтаксисе команд.

- Объединение в одном и том же синтаксисе функциональности всех языков системы БД (языка определения данных, языка манипулирования данными, языка определения ограничений целостности, языка безопасности данных).

QBE – высокоуровневый язык управления базами данных, предоставляющий удобный и унифицированный стиль для построения запросов, обновления, определения и

контроля базы данных. Такой симбиоз открывает дополнительные возможности, немыслимые для других языков. Например, в QBE можно попросить систему указать в каких столбцах каких таблиц есть значение *123*.

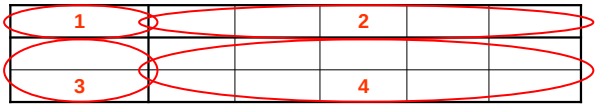
- Мощная и в то же время бесквантерная «человеческая» логика.

Несмотря на то, что QBE относится к классу языков реляционного исчисления с переменными на доменах, пользователь при формулировании запросов явно не оперирует кванторами. Этого удается достичь с использованием других, более понятных человеку конструкций. И вообще, в языковых формах реализованы наиболее близкие именно людям логические выражения из множества всех эквивалентных. Многочисленные примеры продемонстрируют это качество языка.

Согласно философии QBE, от пользователя требуются весьма скромные знания для начала работы, и сводится к минимуму количество концепций, которые он должен изучить для понимания и использования всего языка. Синтаксис языка прост, но, тем не менее, он охватывает широкий спектр сложных операций. Это достигается за счет использования одинаковых операций для извлечения, манипулирования, определения и контроля данных. Операции языка подражают, насколько возможно, ручному манипулированию таблицами, сохраняя простоту и симметричность реляционной модели. Формирование операций следует мыслительному процессу пользователя, предоставляя тем самым свободу при их построении. Система может динамически создавать и уничтожать таблицы базы данных, она также предоставляет пользователю возможности динамического определения предложений контроля данных и средства безопасности.

Архитектура языка QBE направлена на удовлетворение только что сформулированных требований. Результаты различных психологических исследований показывают, что достаточно для инструктирования непрограммистов менее трех часов, после чего они могут формулировать довольно сложные запросы, которые в противном случае требуют от пользователя знания исчисления предикатов первого порядка.

Язык QBE



Области таблицы-шаблона запроса:

1. **Действия с отношениями.**
2. **Действия с атрибутами.**
3. **Действия с кортежами.**
4. **Действия с компонентами кортежей.**

Основная грамматическая конструкция

[<действие>.][<операция сравнения>]
[{<элемент-константа>|<элемент-пример>}]

<действие>::= {I|U|D|P}

В QBE фундаментальными являются две основные концепции. Формулировка команды осуществляется посредством двумерных таблиц-шаблонов. Это достигается заполнением соответствующих полей таблицы примером решения. Проводится также различие между элементом-константой и элементом-примером. Элементы-примеры (переменные) подчеркиваются, а элементы-константы нет. Используя две эти концепции, пользователь может сформулировать широкий спектр запросов.

Запрос на языке QBE формируется заполнением полей одной или нескольких таблиц-шаблонов. В большинстве случаев помещаемые в поля таблицы конструкции имеют следующий синтаксис –

«[<*действие*>.][<*операция сравнения*>][{<*элемент-константа*>|<*элемент-пример*>}]».

В ней можно указать:

- одно из стандартных действий (*I* (от англ. insert) – включение, *U* (от англ. update) – модификация, *D* (от англ. delete) – удаление, *P* (от англ. print) – выборка или печать);
- одну из операций сравнения (=, <>, >, …) (по умолчанию предполагается равенство);
- либо элемент-константу, либо элемент-пример.

В случаях сложных запросов (с квантором всеобщности, с группированием и т.п.) грамматика заполнителей полей может усложняться. В заключение подпункта будут приведены примеры запросов на определение схемы отношения. В них также будет использоваться расширенная грамматика таблиц-шаблонов.

Для запросов на манипулирование структурой отношения и данными все поля таблицы-шаблона разбиваются на группы с одинаковой семантикой. Поля одной группы предполагают манипулирование одними и теми же элементами БД.

1. Левое верхнее поле таблицы-шаблона предназначено для идентификации отношения (таблицы) и действий с ним.
2. Остальные поля заголовка таблицы-шаблона предназначены для идентификации атрибутов отношения (столбцов таблицы) и действий с ними.
3. Поля боковика таблицы-шаблона (за исключением верхнего) предназначены для действий с кортежами отношения (строками таблицы) в целом.
4. Поля тела таблицы-шаблона (за исключением левого столбца) предназначены для действий с отдельными компонентами кортежей отношения (значениями в строках таблицы).

В зависимости от того, где (в поле какой группы таблицы-шаблона) указана основная грамматическая конструкция, она указывает на соответствующий элемент БД или задает определенное действие с ним. Обратите на это внимание при анализе приводимых примеров.

Сначала мы рассмотрим примеры на действие выборки данных, затем поговорим о том, как в QBE осуществляются другие действия с данными, и, наконец, покажем, как в QBE создаются новые структуры, и определяются ограничения целостности для них.

Язык QBE. Диалог с пользователем

P. TAB				

БОЛЬНИЦА P.				

БОЛЬНИЦА	К/Б	Название	Адрес	Ч/К

Как уже отмечалось, ориентация QBE на диалог с пользователем приводит к тому, что система минимизирует усилия пользователя по формулированию запроса.

Допустим, пользователь хочет сформулировать следующий запрос: «*Получить полные сведения о больницах*». Изначально на экране предлагается пустой шаблон таблицы, как показано на слайде.

Пользователь может попросить систему вывести список имен доступных ему таблиц базы данных. Формулировка этого запроса приведена на слайде ниже. Элемент-пример *TAB* является необязательным. Здесь оператор вывода расположен в поле имени таблицы, запрашивая тем самым у системы все доступные имена таблиц. Для нашей медицинской БД результатом этого запроса будет множество таблиц-шаблонов, каждая из которых соответствует одному отношению БД. Среди них будет и таблица-шаблон *БОЛЬНИЦА*.

Вместо ввода заголовков столбцов пользователем система может сгенерировать их автоматически. Как это делается, показано на слайде далее. В этом примере запрашиваются заголовки столбцов таблицы *БОЛЬНИЦА*. Здесь оператор вывода применен ко всей строке заголовков, и в действительности этот пример является сокращенной формой расположения нескольких операторов вывода в полях заголовков столбцов. Автоматическое порождение заголовков столбцов полезно, поскольку оно освобождает пользователя от необходимости запоминания этих заголовков (или нахождения их в каталоге), предотвращая тем самым ошибки при вводе.

Если пользователь введет *P. TAB P.* (или *P. P.*) в поле имени таблицы, система выведет структуру БД, т. е. имена таблиц и имена соответствующих этим таблицам столбцов.

Реляционное исчисление с переменными на доменах. Язык QBE. Примеры запросов

Получить полные сведения о больницах

$\{x_1x_2x_3x_4 \mid БОЛЬНИЦА(x_1x_2x_3x_4)\}$

БОЛЬНИЦА	К/Б	Название	Адрес	Ч/К
	P.X1	P.X2	P.X3	P.X4

БОЛЬНИЦА	К/Б	Название	Адрес	Ч/К
P.				

Получить фамилии хирургов

$\{f \mid \exists x \exists y(ВРАЧ(xyf'ХИРУРГ'))\}$

ВРАЧ	К/В	К/Б	Фамилия	Специальность
			P.F	ХИРУРГ

После этого пользователь формулирует запрос, осуществляя ввод, показанный на слайде. Конструкции *X1*, *X2*, *X3*, *X4*, являющиеся элементами-примерами (переменными на доменах), подчеркнуты и представляют пример возможного ответа. Элементы *X1*, *X2*, *X3*, *X4* необязательно должны содержаться в базе данных. Поскольку элементы-примеры являются произвольными, пользователь может обозначать их как угодно без изменения смысла и результатов запроса. Позднее мы рассмотрим элементы-примеры, используемые для установления связей между двумя или более строками одной таблицы или разными таблицами. Там, где не нужны связи (и каждый элемент-пример представлен в запросе лишь один раз), элементы-примеры можно опускать. Тем самым наш первый запрос может быть эквивалентно переформулирован без указания элементов-примеров.

В нашем случае (требуется выдать кортежи отношения *БОЛЬНИЦА* целиком) запрос может быть еще более кратким, как показано во втором варианте. Здесь оператор вывода *P.* применяется ко всей строке. Если таблица-шаблон содержит множество столбцов, пользователь может вывести все необходимые, удалив предварительно ненужные столбцы и применив оператор *P.* для всей строки, представляющей только оставшиеся столбцы.

После того как запрос сформулирован, пользователь нажимает клавишу *Enter* для получения ответа.

Когда пользователь формулирует запрос в таблице-шаблоне, он воспринимает шаблон запроса как реальную таблицу с теми же заголовками, что и таблица запроса, содержащую реальные данные. Другими словами, для выражения запроса он подражает операциям ручного манипулирования таблицами. Если для выражения запроса пользователю необходимы две или больше таблиц, он может сформировать дополнительные пустые шаблоны (используя специальные функциональные клавиши) и затем ввести информацию в их заголовки.

Мы увлеклись рассказом о языке QBE и прежде времени забыли об абстрактном языке реляционного исчисления с переменными на доменах, а ведь, как ни странно, именно к этому классу языков относится QBE.

Как и для языка ALPHA первые два запроса мы приводим и на абстрактном языке реляционного исчисления с переменными на доменах, и на языке QBE. Они позволяют, во-первых, увидеть их сходство, а, во-вторых, отметить их различия, связанные в основном с тем, что QBE ориентирован на диалог с пользователем и поэтому сводит к минимуму усилия человека по формулированию запроса.

Первый вариант первого запроса на QBE, как видим, является «дословным» переводом запроса на языке реляционного исчисления с переменными на доменах. Здесь в

явной форме указаны те же свободные переменные на доменах x_1, x_2, x_3, x_4 (хоть и в виде элементов-примеров *X1*, *X2*, *X3*, *X4*). Следует отметить, что характерной особенностью свободных переменных на доменах в QBE является наличие в соответствующих полях действия выборки (операции *P*). При этом для краткости запроса элементы-примеры для них могут быть опущены, но в системе и в этом случае предполагаются соответствующие свободные переменные. Даже если, как во втором варианте первого запроса, операция P указана на уровне кортежа, неявно задаются свободные переменные на доменах, количество которых равно количеству столбцов в таблице-шаблоне.

Второй запрос на QBE показывает, что незаполненным полям тела таблицы также неявно соответствуют свои переменные на доменах, только в этом случае предполагается, что в запросе они связаны квантором существования. Как вы догадываетесь, элемент-пример *F* в этом запросе можно было опустить. Кстати, этот запрос на QBE можно представить следующей фразой: «Выдать фамилии, такие, например, как *F*, врачей, которые имеют специальность хирург».

Язык QBE. Примеры запросов (продолжение)

Выдать названия палат больницы с кодом 5, имеющих более 10 коек

ПАЛАТА	К/Б	Н/П	Название	Ч/К
	5		P.X	>10

Выдать фамилии пациентов – женщин до 30 лет или мужчин после 50 лет

ПАЦИЕНТ	Р/Н	Фамилия	Адрес	Д/Р	Пол	НМП
		P.X		>1976	Ж	
		P.Y		<1956	М	

Выдать фамилии пациентов от 30 до 50 лет

ПАЦИЕНТ	Р/Н	Фамилия	Адрес	Д/Р	Пол	НМП
		P.X		<1976		
		X		>1956		

Условия, заданные в одной строке, связываются конъюнкцией. Для дизъюнкции условий их нужно указать в разных строках. Если в двух строках используется один и тот же элемент-пример, между их условиями опять предполагается операция конъюнкции.

Мы видим, что элементы-примеры используются для установления связей между строками в сложных запросах. Если же связи не нужны, и элемент-пример указан в запросе лишь однажды, его для краткости можно опустить.

Порядок строк в запросах несущественен. Благодаря коммутативности конъюнкции и дизъюнкции пользователь может обдумывать и кодировать строки запроса в произвольном порядке.

Язык QBE. Примеры запросов (продолжение)

Выдать фамилии врачей, лечащих пациента с Р/Н 111111

ВРАЧ	К/В	К/Б	Фамилия	Специальность
	Y		P.X	

ВРАЧ-ПАЦИЕНТ	К/В	Р/Н
	Y	111111

Выдать фамилии врачей, не лечащих пациента с Р/Н 111111

ВРАЧ	К/В	К/Б	Фамилия	Специальность
	Y		P.X	

ВРАЧ-ПАЦИЕНТ	К/В	Р/Н
¬	Y	111111

Выдать фамилии врачей, лечащих кого-нибудь, кроме пациента с Р/Н 111111

ВРАЧ	К/В	К/Б	Фамилия	Специальность
	Y		P.X	

ВРАЧ-ПАЦИЕНТ	К/В	Р/Н
	Y	<>111111

В этих случаях пользователь использует и таблицу *ВРАЧ*, и таблицу *ВРАЧ-ПАЦИЕНТ*, порождая два пустых шаблона и заполняя их заголовками и необходимыми элементами запроса. В этих примерах проиллюстрирована важность элементов-примеров. В каждом из запросов должен быть применен один и тот же элемент-пример (Y) в обеих таблицах, показывая, что если некоторое значение *Кода врача* удовлетворяет условию, указанному в таблице-шаблоне *ВРАЧ-ПАЦИЕНТ*, то фамилия врача с этим значением *Кода врача* должна быть выдана в результате.

Вместо словесных описаний этих запросов приведем их формулировки на абстрактном языке реляционного исчисления с переменными на доменах, которые лучше всяких слов пояснят соответствующие конструкции.

Для первого запроса –

$\{x \mid \exists y \exists v \exists w(ВРАЧ(yvxw) \wedge ВРАЧ - ПАЦИЕНТ(y'111111'))\}$.

Для второго запроса –

$\{x \mid \exists y \exists v \exists w(ВРАЧ(yvxw) \wedge \neg ВРАЧ - ПАЦИЕНТ(y'111111'))\}$.

Для третьего запроса –

$\{x \mid \exists y \exists v \exists w \exists z(ВРАЧ(yvxw) \wedge ВРАЧ - ПАЦИЕНТ(yz) \wedge z \neq '111111')\}$.

Поскольку в формулировках запросов нет никаких указаний на то, как должен обрабатываться запрос, и в каком порядке должен осуществляться просмотр таблиц, пользователь волен выбирать порядок задания таблиц-шаблонов. Если к этому добавить безразличие к порядку строк в таблицах-шаблонах, можно сказать, что грамматике языка QBE свойственна «коммутативность» таблиц-шаблонов и их строк. Понятно, что она обязана этим коммутативности логических операций конъюнкции и дизъюнкции.

Язык QBE. Примеры запросов (продолжение)

Выдать коды врачей, лечащих по крайней мере одного больного, что и врач с кодом 999

ВРАЧ-ПАЦИЕНТ	К/В	Р/Н
	P.X	Y
	999	Y

Выдать Р/Н пациентов, которые ходили ко всем врачам

ВРАЧ	К/В	К/Б	Фамилия	Специальность
	ALL.Y			

ВРАЧ-ПАЦИЕНТ	К/В	Р/Н
	ALL.Y	P.X

Первый запрос слайда, по-видимому, в пояснениях не нуждается. Он, как и все предыдущие относится к запросам с квантором существования.

А вот второй запрос имеет скрытый квантор всеобщности. Действительно, его формулировка на абстрактном языке реляционного исчисления с переменными на доменах выглядит следующим образом:

$\{x \mid \forall y(\exists u \exists v \exists w(ВРАЧ(yuvw)) \leftrightarrow ВРАЧ - ПАЦИЕНТ(yx))\}$.

Злуф предлагает подобные запросы трактовать так. Выражение *ALL.*$\underline{Y}$ в таблице *ВРАЧ* определяет множество кодов врачей, таких как $\underline{Y}$, имеющихся в таблице *ВРАЧ*. Второе выражение *ALL.*$\underline{Y}$ в таблице *ВРАЧ-ПАЦИЕНТ* определяет множество всех кодов врачей, образующих строки этой таблицы вместе с текущим проверяемым на предмет выполнения операции печати значением регистрационного номера, таким как $\underline{X}$. Поскольку только эти выражения представлены в соответствующих полях таблиц-шаблонов, и они полностью совпадают, между этими множествами в запросе предполагается отношение равенства. Что фактически и представлено в запросе на абстрактном языке реляционного исчисления с переменными на доменах в виде логической операции эквивалентности.

Язык QBE. Примеры запросов (продолжение)

Выдать коды врачей, лечащих по крайней мере тех же больных, что и врач с кодом 999

ВРАЧ-ПАЦИЕНТ	К/В	Р/Н
	P.X	ALL.Y *
	999	ALL.Y

Последний пример запроса на выборку информации также использует конструкцию «*ALL.*», и, как не трудно догадаться, в нем также имеется скрытый квантор всеобщности. Его формулировка на абстрактном языке реляционного исчисления с переменными на доменах выглядит следующим образом:

$$\{x \mid \forall y(ВРАЧ-ПАЦИЕНТ('999'\, y) \rightarrow ВРАЧ-ПАЦИЕНТ(xy))\}.$$

Как видим, в этом выражении по сравнению с предыдущим примером логическая операция эквивалентности заменена импликацией.

Действительно, выражение *ALL.*$\underline{Y}$ во второй строке таблицы-шаблона определяет множество регистрационных номеров, таких как $\underline{Y}$, образующих вместе с кодом врача, равным *999*, строки таблицы *ВРАЧ-ПАЦИЕНТ*. С другой стороны, конструкция с квадратными скобками в первой строке таблицы-шаблона определяет множество регистрационных номеров, образующих вместе с проверяемым кодом врача, таким как $\underline{X}$, строки той же таблицы. Тот факт, что в квадратных скобках наряду с выражением *ALL.*$\underline{Y}$, идентичным указанному во второй строке, стоит знак «*», предполагающий возможность наличия дополнительных элементов множества, говорит о том, что в запросе предполагается отношение включения между этими множествами.

Как видим, логическое мышление кванторами всеобщности заменено в QBE адекватным теоретико-множественным мышлением отношениями равенства и включения множеств. Первое отношение достигается указанием идентичных конструкций, образующих множества, второе – дополнительным использованием звездочки.

Обобщенное скобочное выражение может содержать одно или более множеств, а также произвольное количество единичных элементов-примеров и элементов-констант. В него также может входить или не входить единичная звездочка.

Язык QBE. Примеры запросов (продолжение). Операции модификации БД

Добавить нового служащего

ПЕРСОНАЛ	К/Б	Н/П	Фамилия	Должность	Смена	З/П
I.	5	2	Смит	Няня	Н	1000

Сделать всему персоналу палаты номер 2 больницы с кодом 5 дневную смену

ПЕРСОНАЛ	К/Б	Н/П	Фамилия	Должность	Смена	З/П
	5	2			U.Д	

Удалить служащих по фамилии Смит

ПЕРСОНАЛ	К/Б	Н/П	Фамилия	Должность	Смена	З/П
D.			Смит			

Удалить всех служащих

ПЕРСОНАЛ	К/Б	Н/П	Фамилия	Должность	Смена	З/П
D.						

На слайде приведены примеры запросов на языке QBE, осуществляющих действия, которые изменяют состояние БД. Примеры достаточно очевидны и в пояснениях не нуждаются.

Язык QBE. Примеры на определение схемы отношения

I. БОЛЬНИЦА I.	К/Б	Название	Адрес	Ч/К

I. БОЛЬНИЦА I.	К/Б	Название	Адрес	Ч/К
P. XX				

БОЛЬНИЦА	К/Б	Название	Адрес	Ч/К
TYPE LENGTH KEY DOMAIN NULL				

БОЛЬНИЦА		К/Б	Название	Адрес	Ч/К
TYPE	I.	INTEGER	CHAR	CHAR	INTEGER
LENGTH	I.	4	40	50	5
KEY	I.	K	NK	NK	NK
DOMAIN	I.	IDENT	NAME	ADDRESS	-
NULL	I.	NOT	NOT	YES	YES

Создание таблицы в языке QBE выполняется в том же стиле, что и описанные ранее операции. Таблица определяется при помощи той же таблицы-шаблона с элементами-константами и элементами-примерами. Пользователю предоставляются средства для создания новых таблиц, изменения структуры существующих таблиц, задания ограничений целостности для таблицы.

На слайде продемонстрировано создание таблицы с именем *БОЛЬНИЦА* и столбцами *К/Б, НАЗВАНИЕ, АДРЕС, Ч/К*. Начав с пустого шаблона (как при формулировании выборки), пользователь заполняет заголовок именем таблицы и именами полей. Оператор *I*. справа от «*БОЛЬНИЦА*» относится ко всей строке заголовков столбцов.

Для задания типов данных, размеров, состояний «ключ-неключ», доменов и обязательности столбцов используются характеристики столбцов. Например, можно одновременно с созданием таблицы запросить имена характеристик столбцов, как это показано во втором запросе. Данный запрос создает новую таблицу и затем выполняет вывод в боковике таблицы-шаблона всех имен характеристик столбцов в системе. Результат этого запроса показан в виде третьей таблицы. Пользователь затем определяет ограничения целостности для таблицы, заполняя строки шаблона, как показано в последней таблице.

Ниже приведены описания характеристик столбцов для таблиц.

- *TYPE* задает тип значений столбца, например *CHAR, INTEGER* и т.д.
- *LENGTH* задает максимальную длину значений столбца.
- *KEY* используется для указания столбцов, которые будут рассматриваться как поля первичного ключа. *K* означает ключ, *NK* означает не ключ.
- *DOMAIN* задает имя домена, т.е. множества значений, которому должны принадлежать значения столбца.
- *NULL* определяет обязательность (*NOT*) и необязательность (*YES*) значений столбца в строках таблицы.

Язык QBE. Примеры на определение схемы отношения (продолжение)

БОЛЬНИЦА	К/Б	Название	Адрес	Ч/К
P. XX P.				

БОЛЬНИЦА	К/Б	Название	Адрес	Ч/К	I. УРОВЕНЬ
TYPE	INTEGER	CHAR	CHAR	INTEGER	I. CHAR
LENGTH	4	40	50	5	I. 20
KEY	K	NK	NK	NK	I. NK
DOMAIN	IDENT	NAME	ADDRESS	-	I. ATO_TYPE
NULL	NOT	NOT	YES	YES	I. YES

Владелец таблицы может расширить ее определение, действуя примерно так же, как при создании новой таблицы. Например, он может добавить к таблице *БОЛЬНИЦА* столбец *УРОВЕНЬ*, значения которого отражают уровень больницы (федеральный, региональный, местный). Сначала пользователь должен запросить для таблицы *БОЛЬНИЦА* имена характеристик столбцов и их значения для существующих столбцов таблицы (первый запрос). Результатом этого запроса будет полная схема таблицы, как она была определена ранее. Пользователь должен вставить имя нового столбца и ввести для него значения характеристик, как показано во втором запросе. Если в таблице есть данные, то уже существующие строки будут содержать в поле *БОЛЬНИЦА NULL*-значения, пока пользователь не переопределит их.

Оператор *U.* изменяет схему таблицы – будь то заголовки или характеристики – подобно тому, как при помощи оператора *I.* осуществляется вставка. Например, если нужно изменить имя таблицы с *БОЛЬНИЦА* на *КЛИНИКА*, то достаточно ввести перед именем таблицы оператор *U.* и напечатать новое имя вместо старого. При нажатии клавиши *Enter* имя таблицы будет изменено.

При помощи оператора *D.* можно уничтожать элементы схемы точно так же, как обычные данные. Например, задав оператор *D.* в строке, содержащей соответствие столбцов доменам, вы уничтожите соответствующие спецификации.

Для того, чтобы уничтожить столбец таблицы, достаточно указать перед его именем оператор *D.* Фактически такой оператор – это сокращенный способ выполнить удаление всех данных столбца и самого столбца. Аналогично, указание оператора *D.* перед именем таблицы – это быстрый способ для уничтожения данных, ограничений целостности и структуры таблицы.

Мы привели обзор средств языка QBE для выполнения выборки, манипулирования и определения данных. Уникальное свойство языка заключается в том, что для его первоначального понимания и использования достаточно освоить лишь минимальное число основных понятий.

В противоположность англоподобным языкам с линейным синтаксисом, где пользователь должен придерживаться определенной структуры предложений, в среде QBE можно ввести в качестве элемента шаблона любое выражение, если только оно синтаксически корректно. Иначе говоря, поскольку элементы привязаны к шаблонам таблиц, то пользователь имеет возможность вводить лишь допустимые запросы. В языке запросов англоподобной структуры всегда есть опасность ввести запрос, не удовлетворяющий синтаксическим правилам.

Поскольку порядок заполнения таблиц неважен, то при формулировании операции предоставляется множество степеней свободы.

Пользователь может строить запрос, постепенно доопределяя новые условия; таким образом, можно постепенно переходить от простых запросов к более сложным.

Операции выборки, манипулирования, определения данных выполняются единообразно, с минимальными различиями в синтаксических правилах.

4.2.4.4. Язык SQL

Одним из языков, появившихся в результате разработки реляционной модели данных, является **Structured Query Language (SQL)**, который в настоящее время получил очень широкое распространение и фактически превратился в стандартный язык реляционных баз данных. Стандарт на язык SQL был выпущен Национальным институтом стандартизации США (ANSI) в 1986 году, а в 1987 году Международная организация по стандартизации (ISO) приняла этот стандарт в качестве международного. В настоящее время язык SQL поддерживается сотнями СУБД различных типов, разработанных для самых разнообразных вычислительных платформ, начиная от персональных компьютеров и заканчивая мэйнфреймами.

История реляционной модели данных (и косвенно языка SQL) началась в 1970 году с публикации основополагающей статьи Э. Ф. Кодда (в то время он работал в исследовательской лаборатории корпорации IBM в Сан-Хосе). В 1974 году Д. Чемберлен, работавший в той же лаборатории, публикует определение языка, получившего название «Structured English Query Language» или SEQUEL. В 1976 году была выпущена переработанная версия этого языка SEQUEL/2. Впоследствии его название пришлось изменить на SQL по юридическим соображениям – аббревиатура SEQUEL уже использовалась кем-то ранее. Но до настоящего времени многие по-прежнему произносят аббревиатуру SQL как «сиквэл», хотя официально ее рекомендуется читать как «эс-кю-эл».

В 1976 году на базе языка SEQUEL/2 корпорация IBM выпустила прототип СУБД, имевший название System R. Назначение этой пробной версии состояло в проверке осуществимости реляционной модели. Помимо прочих положительных аспектов, важнейшим из результатов выполнения этого проекта можно считать разработку собственно языка SQL. Однако корни этого языка уходят в язык SQUARE (Specifying Queries as Rational Expressions), который являлся предшественником проекта System R. Язык SQUARE был разработан как исследовательский инструмент для реализации реляционной алгебры посредством фраз, составленных на английском языке.

В конце 1970-х годов, компанией, которая ныне превратилась в корпорацию Oracle, была выпущена СУБД Oracle. Пожалуй, это самая первая из коммерческих реализаций реляционной СУБД, построенной на использовании языка SQL. В 1981 году корпорация IBM выпустила свою первую коммерческую реляционную СУБД под названием SQL/DS (для среды DOS/VSE). В 1982 году вышла в свет версия этой системы для среды VM/CMS, а в 1983 году – для среды MVS, но уже под названием DB2.

В 1982 году Национальный институт стандартизации США начал работу над языком RDL (Relation Database Language), руководствуясь концептуальными документами, полученными от корпорации IBM. В 1983 году к этой работе подключилась Международная организация по стандартизации. Совместные усилия обеих организаций увенчались выпуском стандарта языка SQL. От названия RDL в 1984 году отказались, а черновой проект языка был переработан с целью приближения к уже существующим реализациям языка SQL.

Исходный вариант стандарта, который был выпущен ISO в 1987 году, вызвал волну критических замечаний. В частности, Дейт, известный исследователь в этой области, указывал, что в стандарте опущены важнейшие функции, включая средства обеспечения ссылочной целостности и некоторые реляционные операторы. Кроме того, он отметил чрезмерную избыточность языка – один и тот же запрос можно было записать в нескольких различных вариантах. Большая часть критических замечаний была признана справедливой, и необходимые коррективы были внесены в стандарт еще до его публикации. Однако было решено, что важнее выпустить стандарт как можно быстрее (чтобы он смог исполнять роль общей основы, на которой и сам язык, и его реализации

могли бы развиваться далее), чем дожидаться, пока будут определены и согласованы все функции, которые разные специалисты считают обязательными для подобного языка.

В 1989 году ISO опубликовала дополнение к стандарту, в котором определялись функции поддержки целостности данных. В 1992 году была выпущена первая, существенно пересмотренная версия стандарта ISO, которую иногда называют SQL2 (или SQL-92). Хотя некоторые из функций были определены в этом стандарте впервые, многие из них уже были полностью или частично реализованы в одной или нескольких коммерческих реализациях языка SQL.

А следующая версия стандарта, которую принято называть SQL3, была выпущена только в 1999 году. Эта версия содержит дополнительные средства поддержки объектно-ориентированных функций управления данными.

Функции, которые добавляются к стандарту языка разработчиками коммерческих реализаций, принято называть расширениями. Например, в стандарте языка SQL определено шесть различных типов данных, которые могут храниться в базах данных. Во многих реализациях этот список дополняется разнообразными расширениями. Каждая из реализаций языка называется диалектом. Не существует двух совершенно идентичных диалектов, как в настоящее время не существует и ни одного диалекта, полностью соответствующего стандарту ISO.

Более того, поскольку разработчики СУБД вводят в системы все новые функциональные средства, они постоянно расширяют свои диалекты языка SQL, в результате чего отдельные диалекты все больше и больше отличаются друг от друга. Однако основное ядро языка SQL остается более или менее стандартизованным во всех реализациях.

Мы рассмотрим сначала конструкции ядра языка, заложенные стандартом SQL/89, а затем перейдем к рассмотрению диалекта языка, реализованного в СУБД Oracle.

Стандарт SQL. Конструкции запросов

```
<cursor specification> ::= <query expression> [<order by clause>]
<query expression> ::= <query term> | <query expression> {UNION
                       [ALL] | INTERSECT | MINUS} <query term>
<query term> ::= <query specification> | (<query expression>)
<query specification> ::= (SELECT [ALL | DISTINCT] <select list>
                           <table expression>)

<select statement> ::= SELECT [ALL | DISTINCT] <select list> INTO
                           <select target list> <table expression>

<subquery> ::= (SELECT [ALL | DISTINCT] <result specification>
                           <table expression>)

<table expression> ::= <from clause> [<where clause>]
                           [<group by clause>] [<having clause>]
```

Язык допускает три типа синтаксических конструкций, начинающихся с ключевого слова *SELECT*: спецификация курсора (*cursor specification*), оператор выборки (*select statement*) и подзапрос (*subquery*). В основе каждой из них лежит синтаксическая конструкция «табличное выражение (*table expression*)». Семантика табличного выражения состоит в том, что на основе последовательного применения разделов *from*, *where*, *group by* и *having* из заданных в разделе *from* таблиц строится некоторая новая результирующая таблица, порядок следования строк которой не определен и среди строк которой могут находиться дубликаты (т.е. в общем случае таблица-результат табличного выражения является мультимножеством строк).

СПЕЦИФИКАЦИЯ КУРСОРА

Наиболее общей является конструкция «спецификация курсора». Курсор – это средство языка SQL, позволяющее с помощью набора специальных операторов получить построчный доступ к результату запроса к БД. На интерактивные запросы также распространяется грамматика спецификации курсора. К табличным выражениям, участвующим в спецификации курсора, не предъявляются какие-либо ограничения. Как видно из сводки синтаксических правил, при определении спецификации курсора используются три дополнительных конструкции: спецификация запроса (*query specification*), выражение запроса (*query expression*) и раздел *ORDER BY* (*order by clause*).

Спецификация запроса

В спецификации запроса задается список выборки (список выражений над значениями столбцов результата табличного выражения и константами). В результате применения списка выборки к результату табличного выражения производится построение новой таблицы, содержащей то же число строк, но, вообще говоря, другое число столбцов, значения которых формируются на основе вычисления соответствующих выражений из списка выборки. Кроме того, в спецификации запроса могут содержаться ключевые слова *ALL* или *DISTINCT*. При наличии ключевого слова *DISTINCT* из таблицы, полученной применением списка выборки к результату табличного выражения, удаляются строки-дубликаты; при указании *ALL* (или просто при отсутствии *DISTINCT*) удаление строк-дубликатов не производится.

Выражение запроса

Выражение запроса – это выражение, строящееся по указанным синтаксическим правилам на основе спецификаций запросов. Единственной операцией, которую разрешается использовать в выражениях запросов SQL/89, является операция *UNION* (объединение таблиц) с возможной разновидностью *UNION ALL*. К таблицам-операндам выражения запроса предъявляется то требование, что все они должны содержать одно и то

же число столбцов, и соответствующие столбцы всех операндов должны быть одного и того же типа. Выражение запроса вычисляется слева направо с учетом скобок. При выполнении операции *UNION* производится обычное теоретико-множественное объединение операндов, т.е. из результирующей таблицы удаляются дубликаты. При выполнении операции *UNION ALL* образуется результирующая таблица, в которой могут содержаться строки-дубликаты. Позже в стандартный синтаксис были добавлены теоретико-множественные операции пересечение (*INTERSECT*) и разность (*MINUS*).

Раздел *ORDER BY*

Наконец, раздел *ORDER BY* позволяет установить желаемый порядок просмотра результата выражения запросов. Синтаксис *ORDER BY* следующий:

<order by clause> ::= ORDER BY <sort specification> [{,<sort specification>}...]

<sort specification> ::= {<unsigned integer> | <column specification>} [ASC | DESC]

Как видно из этих синтаксических правил, фактически задается список столбцов результата выражения запроса, и для каждого столбца указывается порядок просмотра строк результата в зависимости от значений этого столбца (*ASC* – по возрастанию (умолчание), *DESC* – по убыванию). Столбцы можно задавать их именами тогда и только тогда, когда (1) выражение запроса не содержит операций *UNION* или *UNION ALL* и (2) в списке выборки спецификации запроса этому столбцу соответствует выражение, состоящее только из имени столбца. Во всех остальных случаях в разделе *ORDER BY* должен указываться порядковый номер столбца в таблице-результате выражения запроса.

ОПЕРАТОР ВЫБОРКИ

Оператор выборки – это отдельный оператор языка SQL/89, позволяющий получить результат запроса в прикладной программе без использования курсора. Поэтому оператор выборки имеет синтаксис, отличающийся от синтаксиса спецификации курсора, и при выполнении оператора возникают ограничения на результат табличного выражения. Фактически, и то и другое диктуется спецификой оператора выборки как одиночного оператора SQL – при его выполнении результат должен быть помещен в переменные прикладной программы. Поэтому в операторе появляется раздел *INTO*, содержащий список переменных прикладной программы, и возникает то ограничение, что результирующая таблица должна содержать не более одной строки. Соответственно, результат базового табличного выражения должен содержать не более одной строки, если оператор выборки не содержит спецификации *DISTINCT*, и таблица, полученная применением списка выборки к результату табличного выражения, должна состоять только из строк-дубликатов, если спецификация *DISTINCT* задана.

ПОДЗАПРОС

Наконец, последняя конструкция SQL/89, которая может содержать табличные выражения, – это подзапрос, т.е. запрос, который может входить в предикат условия выборки оператора SQL. В SQL/89 к подзапросам применяется то ограничение, что результирующая таблица должна содержать в точности один столбец. Поэтому в синтаксических правилах, определяющих подзапрос, вместо списка выборки указано «выражение, вычисляющее значение (*result specification*)». (Следует отметить, что в диалекте Oracle это ограничение отсутствует.) Заметим еще, что поскольку подзапрос всегда вложен в некоторый другой оператор SQL, то вместо констант в выражении выборки и логических выражениях разделов *WHERE* и *HAVING* можно использовать значения столбцов текущих строк таблиц, участвующих во внешних запросах.

ТАБЛИЧНОЕ ВЫРАЖЕНИЕ

Стандарт SQL/89 рекомендует рассматривать вычисление табличного выражения как последовательное применение разделов *FROM, WHERE, GROUP BY* и *HAVING* к таблицам, заданным в списке *FROM.*

Стандарт SQL. Конструкции табличного выражения

```
<from clause> ::= FROM <table reference> [{,<table reference>}...]
<table reference> ::= <table name> [<correlation name>]

<where clause> ::= WHERE <search condition>
<search condition> ::= <boolean term> | <search condition> OR
                       <boolean term>
<boolean term> ::= <boolean factor> | <boolean term> AND
                       <boolean factor>
<boolean factor> ::= [NOT] <boolean primary>
<boolean primary> ::= <predicate> | (<search condition>)

<comparison predicate> ::= <value expression> <comp op>
                       {<value expression> | <subquery>}
<comp op> ::= = | <> | < | > | <= | >=

<between predicate> ::= <value expression> [NOT] BETWEEN
                   <value expression> AND <value expression>
```

РАЗДЕЛ *FROM*

Раздел *FROM* имеет следующий синтаксис:

<from clause> ::= FROM <table reference> [{,<table reference>}...]

<table reference> ::= <table name> [<correlation name>]

Результатом выполнения раздела *FROM* является расширенное декартово произведение таблиц, заданных списком таблиц раздела *FROM*. Расширенное декартово произведение (расширенное, потому что в качестве операндов и результата допускаются мультимножества) в стандарте определяется следующим образом:

«Расширенное произведение *R* есть мультимножество всех строк *r*, таких, что *r* является конкатенацией строк из всех идентифицированных таблиц в том порядке, в котором они идентифицированы. Мощность *R* есть произведение мощностей идентифицированных таблиц. Порядковый номер столбца в *R* есть *n*+*s*, где *n* – порядковый номер порождающего столбца в именованной таблице *T*, а *s* – сумма степеней всех таблиц, идентифицированных до *T* в разделе *FROM*».

Как видно из синтаксиса, рядом с именем таблицы можно указывать еще одно имя «*correlation name*». Фактически, это некоторый синоним (иногда его называют алиасом) имени таблицы, который можно использовать в других разделах табличного выражения для ссылки на строки именно этого вхождения таблицы. (Одна и та же таблица может участвовать несколько раз в списке одного раздела *FROM* и/или входить в списки разделов *FROM* нескольких (под-)запросов.)

Если табличное выражение содержит только раздел *FROM* (это единственный обязательный раздел табличного выражения), то результат табличного выражения совпадает с результатом раздела *FROM*.

РАЗДЕЛ *WHERE*

Если в табличном выражении присутствует раздел *WHERE*, то далее вычисляется он. Синтаксис раздела *WHERE* следующий:

<where clause> ::= WHERE <search condition>

<search condition> ::= <boolean term> | <search condition> OR <boolean term>

<boolean term> ::= <boolean factor> | <boolean term> AND <boolean factor>

<boolean factor> ::= [NOT] <boolean primary>

<boolean primary> ::= <predicate> | (<search condition>)

Вычисление раздела *WHERE* производится по следующим правилам: пусть *R* – результат вычисления раздела *FROM*. Тогда условие поиска (*search condition*) применяется ко всем строкам *R*, и результатом раздела *WHERE* является таблица, состоящая из тех строк *R*, для которых результатом вычисления условия поиска является *true*. Если условие выборки включает подзапросы, то каждый подзапрос вычисляется для

каждого кортежа таблицы *R* (в стандарте используется термин «effectively» в том смысле, что результат должен быть таким, как если бы каждый подзапрос действительно вычислялся заново для каждого кортежа *R*, хотя реально это требуется далеко не всегда).

Заметим, что поскольку SQL/89 допускает наличие в базе данных неопределенных значений, то вычисление условия поиска должно производиться не в булевой, а в трехзначной логике со значениями *true, false* и *unknown* (неизвестно). Для любого предиката известно, в каких ситуациях он может порождать значение *unknown*. Булевские операции *AND, OR* и *NOT* работают в трехзначной логике следующим образом:

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

Значение выражения не определено, если в его вычислении участвует хотя бы одно неопределенное значение. Еще одно важное замечание из стандарта SQL/89: в контексте *GROUP BY, DISTINCT* и *ORDER BY* неопределенное значение выступает как специальный вид определенного значения, т.е. возможно, например, образование группы строк, значение указанного столбца которых является неопределенным.

Предикатами условия поиска в соответствии с SQL/89 могут быть следующие предикаты: предикат сравнения, предикат *between*, предикат *in*, предикат *like*, предикат *null*, предикат с квантором и предикат *exists*.

Предикат сравнения

Синтаксис предиката сравнения определяется следующими правилами:

<comparison predicate> ::= <value expression> <comp op>
{<value expression> | <subquery>}
<comp op> ::= = | <> | < | > | <= | >=

Через «<>» обозначается операция неравенства. Выражения левой и правой частей предиката сравнения строятся по общим правилам построения выражений и могут включать в общем случае имена столбцов таблиц из раздела *FROM* и константы (не обязательно литеральные; вместо литеральной константы может использоваться имя столбца таблицы, указанной в разделе *FROM* внешнего подзапроса, или имя переменной программы, написанной на объемлющем языке). Типы данных выражений должны быть сравнимыми (например, если тип столбца *a* таблицы *A* является типом символьных строк, то предикат «*a* = 5» недопустим).

Если правый операнд операции сравнения задается подзапросом, то дополнительным ограничением является то, что мощность результата подзапроса должна быть не более единицы. Если хотя бы один из операндов операции сравнения имеет неопределенное значение или если правый операнд является подзапросом с пустым результатом, то значение предиката сравнения равняется *unknown*.

Предикат *between*

Предикат *between* имеет следующий синтаксис:

<between predicate> ::= <value expression> [NOT] BETWEEN <value expression>
AND <value expression>

По определению результат «*x BETWEEN y AND z*» тот же самый, что результат логического выражения «*x >= y AND x <= z*». Результат «*x NOT BETWEEN y AND z*» тот же самый, что результат «NOT *(x BETWEEN y AND z)*».

Стандарт SQL. Конструкции табличного выражения (продолжение)

```
<in predicate> ::= <value expression> [NOT] IN
                        {<subquery> | (<in value list>)}
<in value list> ::= <value specification> {,<value specification>}...

<like predicate> ::= <column specification> [NOT] LIKE
                        <pattern> [ESCAPE <escape character>]
<pattern> ::= <value specification>
<escape character> ::= <value specification>

<null predicate> ::= <column specification> IS [NOT] NULL

<quantified predicate> ::= <value expression> <comp op> <quantifier>
                                <subquery>
<quantifier> ::= <all> | <some>
<all> ::= ALL
<some> ::= SOME | ANY

<exists predicate> ::= EXISTS <subquery>
```

Предикат *in*

Предикат *in* определяется следующими синтаксическими правилами:

<in predicate> ::= <value expression> [NOT] IN {<subquery> | (<in value list>)}

<in value list> ::= <value specification> {,<value specification>}...

Типы левого операнда и значений из списка правого операнда (напомним, что результирующая таблица подзапроса должна содержать ровно один столбец) должны быть сравнимыми.

Значение предиката равно *true* в том и только том случае, когда значение левого операнда совпадает хотя бы с одним значением списка правого операнда. Если список правого операнда пуст (так может быть, если правый операнд задается подзапросом) или значение «подразумеваемого» предиката сравнения $x = y$ (где x – значение выражения левого операнда) равно *false* для каждого элемента y списка правого операнда, то значение предиката *in* равно *false*. В противном случае значение предиката *in* равно *unknown* (например, так может быть, если значение левого операнда есть *NULL*). По определению значение предиката «*x NOT IN S*» равно значению предиката «*NOT (x IN S)*».

Предикат *like*

Предикат *like* имеет следующий синтаксис:

<like predicate> ::= <column specification> [NOT] LIKE <pattern>
[ESCAPE <escape character>]

<pattern> ::= <value specification>

<escape character> ::= <value specification>

Типы данных столбца левого операнда и образца должны быть типами символьных строк. В разделе *ESCAPE* должен специфицироваться одиночный символ.

Значение предиката равно *true*, если значение указанного столбца (*column specification*) удовлетворяет заданному шаблону (*pattern*). При этом если раздел *ESCAPE* отсутствует, то при сопоставлении шаблона со строкой производится специальная интерпретация двух символов шаблона: символ подчеркивания («_») обозначает любой одиночный символ; символ процента («%») обозначает последовательность произвольных символов произвольной длины (может быть, нулевой).

Если же раздел *ESCAPE* присутствует и специфицирует некоторый одиночный символ *x*, то пары символов шаблона «*x*_» и «*x*%» представляют одиночные символы «_» и «%», соответственно.

Значение предиката *like* есть *unknown*, если значение столбца либо шаблона не определено.

Значение предиката «*x NOT LIKE y ESCAPE z*» совпадает со значением «*NOT (x LIKE y ESCAPE z)*».

Предикат *null*

Предикат *null* описывается синтаксическим правилом

<null predicate> ::= <column specification> IS [NOT] NULL

Этот предикат всегда принимает значения *true* или *false*. При этом значение «*x IS NULL*» равно *true* тогда и только тогда, когда значение *x* не определено. Значение предиката «*x IS NOT NULL*» равно значению «*NOT (x IS NULL)*».

Предикат с квантором

Предикат с квантором имеет следующий синтаксис:

<quantified predicate> ::= <value expression> <comp op> <quantifier> <subquery>

<quantifier> ::= <all> | <some>

<all> ::= ALL

<some> ::= SOME | ANY

Обозначим через *x* результат вычисления выражения левой части предиката, а через *S* результат вычисления подзапроса.

Предикат «*x <comp op> ALL S*» имеет значение *true*, если *S* пусто или значение предиката «*x <comp op> s*» равно *true* для каждого *s*, входящего в *S*. Предикат «*x <comp op> ALL S*» имеет значение *false*, если значение предиката «*x <comp op> s*» равно *false* хотя бы для одного *s*, входящего в *S*. В остальных случаях значение предиката «*x <comp op> ALL S*» равно *unknown*.

Предикат «*x <comp op> SOME S*» имеет значение *false*, если *S* пусто или значение предиката «*x <comp op> s*» равно *false* для каждого *s*, входящего в *S*. Предикат «*x <comp op> SOME S*» имеет значение *true*, если значение предиката «*x <comp op> s*» равно *true* хотя бы для одного *s*, входящего в *S*. В остальных случаях значение предиката «*x <comp op> SOME S*» равно *unknown*.

Предикат *exists*

Предикат *exists* имеет следующий синтаксис:

<exists predicate> ::= EXISTS <subquery>

Значением этого предиката всегда является *true* или *false*, и это значение равно *true* тогда и только тогда, когда результат вычисления подзапроса не пуст.

Стандарт SQL. Конструкции табличного выражения (продолжение)

```
<group by clause> ::= GROUP BY <column specification>
                      [{,<column specification>}...]

<having clause> ::= HAVING <search condition>
```

Агрегатные функции

```
<set function specification> ::= COUNT(*) | <distinct set function> |
                                 <all set function>
<distinct set function> ::= { AVG | MAX | MIN | SUM | COUNT }
                            (DISTINCT <column specification>)
<all set function> ::= { AVG | MAX | MIN | SUM } ([ALL]
                       <value expression>)
```

РАЗДЕЛ *GROUP BY*

Если в табличном выражении присутствует раздел *GROUP BY*, то далее выполняется он. Синтаксис раздела *GROUP BY* следующий:

<group by clause> ::= GROUP BY <column specification>
[{,<column specification>}...]

Если обозначить через *R* таблицу, являющуюся результатом предыдущего раздела (*FROM* или *WHERE*), то результатом раздела *GROUP BY* является разбиение *R* на множество групп строк, состоящее из минимального числа таких групп, в которых для каждого столбца из списка столбцов раздела *GROUP BY* во всех строках каждой группы, включающей более одной строки, значения этого столбца совпадают. Для обозначения результата раздела *GROUP BY* в стандарте используется термин «сгруппированная таблица».

РАЗДЕЛ *HAVING*

Наконец, последним при вычислении табличного выражения выполняется раздел *HAVING* (если он присутствует). Синтаксис этого раздела следующий:

<having clause> ::= HAVING <search condition>

Раздел *HAVING* может осмысленно появиться в табличном выражении в том случае, когда в нем присутствует раздел *GROUP BY*. Условие поиска этого раздела задает условие на группу строк сгруппированной таблицы. Раздел *HAVING* может присутствовать и в табличном выражении, не содержащем *GROUP BY*. В этом случае полагается, что результат вычисления предыдущих разделов представляет собой сгруппированную таблицу (правильнее сказать, псевдосгруппированную), состоящую из одной группы без выделенных столбцов группирования.

Условие поиска раздела *HAVING* строится по тем же синтаксическим правилам, что и условие поиска раздела *WHERE*, и может включать те же самые предикаты. Однако имеются специальные синтаксические ограничения по части использования в условии поиска спецификаций столбцов таблиц из раздела *FROM* данного табличного выражения. Эти ограничения следуют из того, что условие поиска раздела *HAVING* задает условие на целую группу, а не на индивидуальные строки.

Поэтому в выражениях предикатов, входящих в условие выборки раздела *HAVING*, прямо можно использовать только спецификации столбцов, указанных в качестве столбцов группирования в разделе *GROUP BY*. Остальные столбцы можно использовать только внутри спецификаций агрегатных функций *COUNT*, *SUM*, *AVG*, *MIN* и *MAX*, вычисляющих в данном случае некоторое агрегатное значение для всей группы строк. Аналогично обстоит дело с подзапросами, входящими в предикаты условия выборки

раздела *HAVING*: если в подзапросе используется характеристика текущей группы, то она может задаваться только путем ссылки на столбцы группирования.

Результатом выполнения раздела *HAVING* является сгруппированная таблица, содержащая только те группы строк, для которых результат вычисления условия поиска есть *true*. В частности, если раздел *HAVING* присутствует в табличном выражении, не содержащем *GROUP BY*, то результатом его выполнения будет либо пустая таблица, либо результат выполнения предыдущих разделов табличного выражения, рассматриваемый как одна группа без столбцов группирования.

АГРЕГАТНЫЕ ФУНКЦИИ И РЕЗУЛЬТАТЫ ЗАПРОСОВ

Агрегатные функции (в стандарте SQL/89 они называются функциями над множествами) определяются следующими синтаксическими правилами:

<set function specification> ::= COUNT(*) | <distinct set function> | <all set function>

<distinct set function> ::= { AVG | MAX | MIN | SUM | COUNT }
(DISTINCT <column specification>)

<all set function> ::= { AVG | MAX | MIN | SUM } ([ALL] <value expression>)

Как видно из этих правил, в стандарте SQL/89 определены пять стандартных агрегатных функций: *COUNT* – число строк или значений, *MAX* – максимальное значение, *MIN* – минимальное значение, *SUM* – суммарное значение и *AVG* – среднее значение.

Семантика агрегатных функций

Агрегатные функции предназначены для того, чтобы вычислять некоторое значение для заданного множества строк. Таким множеством строк может быть группа строк, если агрегатная функция применяется к сгруппированной таблице, или вся таблица. Для всех агрегатных функций, кроме *COUNT*(*), фактический (т.е. требуемый семантикой) порядок вычислений следующий. На основании параметров агрегатной функции из заданного множества строк производится список значений. Затем по этому списку значений производится вычисление функции. Если список оказался пустым, то значение функции *COUNT* для него есть 0, а значение всех остальных функций *null*.

Вычисление функции *COUNT*(*) производится путем подсчета числа строк в заданном множестве. Все строки считаются различными, даже если они состоят из одного столбца со значением *null* во всех строках.

Если агрегатная функция специфицирована с ключевым словом *DISTINCT*, то список значений строится из значений указанного столбца. (Подчеркнем, что в этом случае не допускаются выражения.) Далее из этого списка удаляются неопределенные значения, и в нем устраняются значения-дубликаты. Затем вычисляется указанная функция.

Если агрегатная функция специфицирована без ключевого слова *DISTINCT* (или с ключевым словом *ALL*), то список значений формируется из значений выражения, вычисляемого для каждой строки заданного множества. Далее из списка удаляются неопределенные значения, и производится вычисление агрегатной функции. Обратите внимание, что в этом случае не допускается применение функции *COUNT*.

Результаты запросов

Агрегатные функции можно разумно использовать в спецификации курсора, операторе выборки и подзапросе после ключевого слова *SELECT* (будем называть в этом подразделе все такие конструкции списком выборки, не забывая о том, что в случае подзапроса этот список состоит только из одного элемента), и в условии выборки раздела *HAVING*. Стандарт допускает более экзотические случаи использования агрегатных функций в подзапросах (агрегатная функция на группе кортежей внешнего запроса), но на практике они встречаются очень редко.

Рассмотрим различные случаи применения агрегатных функций в списке выборки в зависимости от вида табличного выражения.

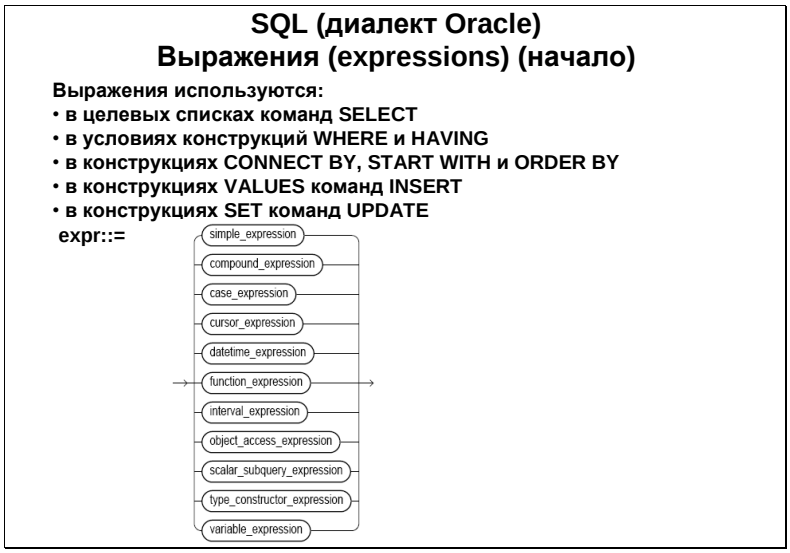
Если результат табличного выражения *R* не является сгруппированной таблицей, то появление хотя бы одной агрегатной функции от множества строк *R* в списке выборки

приводит к тому, что *R* неявно рассматривается как сгруппированная таблица, состоящая из одной (или нуля) групп с отсутствующими столбцами группирования. Поэтому в этом случае в списке выборки не допускается прямое использование спецификаций столбцов *R*: все они должны находиться внутри спецификаций агрегатных функций. Результатом запроса является таблица, состоящая не более чем из одной строки, полученной путем применения агрегатных функций к *R*.

Аналогично обстоит дело в том случае, когда *R* представляет собой сгруппированную таблицу, но табличное выражение не содержит раздела *GROUP BY* (и, следовательно, содержит раздел *HAVING*). Если в предыдущем случае существовало два варианта формирования списка выборки: только с прямым указанием столбцов *R* или только с указанием их внутри спецификаций агрегатных функций, то в данном случае возможен только второй вариант. Результат табличного выражения явно объявлен сгруппированной таблицей, состоящей из одной группы, и результат запроса можно формировать только путем применения агрегатных функций к этой группе строк. Опять результатом запроса является таблица, состоящая не более чем из одной строки, полученной путем применения агрегатных функций к *R*.

Наконец, рассмотрим случай, когда *R* представляет собой «настоящую» сгруппированную таблицу, т.е. табличное выражение содержит раздел *GROUP BY* и, следовательно, определен по крайней мере один столбец группирования. В этом случае правила формирования списка выборки полностью соответствуют правилам формирования условия выборки раздела *HAVING*: допускается прямое использование имен столбцов группирования, а имена остальных столбцов *R* могут появляться только внутри спецификаций агрегатных функций. Результатом запроса является таблица, число строк в которой равно числу групп в *R*, и каждая строка формируется на основе значений столбцов группирования и агрегатных функций для данной группы.

На этом закончим рассмотрение базовых конструкций запросов на языке SQL, введенных самым первым стандартом языка и реализованных в таком виде практически во всех реляционных СУБД, которые поддерживают этот язык. Далее нас ожидает знакомство с конкретным диалектом SQL на примере СУБД Oracle. При этом, как и ранее, оставим без внимания конструкции этого диалекта, относящиеся к объектным расширениям модели данных Oracle, поскольку нас интересует реализация реляционной модели в этой СУБД.



Прежде чем рассматривать команды SQL СУБД Oracle, следует рассказать о базовых конструкциях языка – выражениях и условиях, – используемых в этих командах.

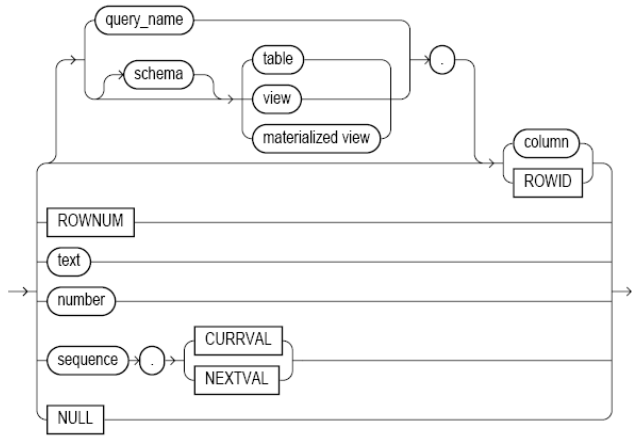
Выражение (англ. expression) – комбинация одного или нескольких значений, операторов и SQL-функций, задающая вычисление некоторого значения. Они могут использоваться:

- в целевых списках команд *SELECT*;
- в условиях конструкций *WHERE* и *HAVING*;
- в конструкциях *CONNECT BY*, *START WITH* и *ORDER BY*;
- в конструкциях *VALUES* команд *INSERT*;
- в конструкциях *SET* команд *UPDATE*.

На слайде перечислены все виды грамматических конструкций выражений. Мы рассмотрим из них лишь наиболее простые и часто используемые.

Выражения (expressions) (продолжение)

simple_expression::=

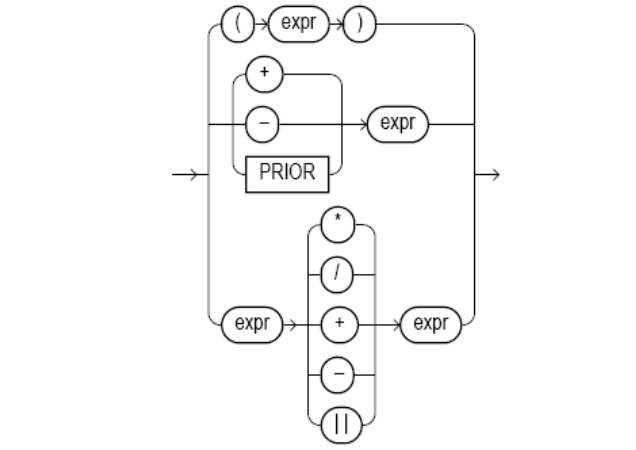


Простое выражение состоит из указания столбца, псевдостолбца, константы, обращения к последовательности или неопределенного значения. В качестве псевдостолбцов могут быть указаны:

- *ROWID* – для каждой строки запроса возвращает физический адрес строки таблицы (*ROWID*);
- *ROWNUM* – для каждой строки запроса возвращает ее порядковый номер в результирующей таблице;
- *LEVEL* – для каждой строки иерархических запросов *LEVEL* = 1 для корневых строк, *LEVEL* = 2 для детей корневых строк и т.д.

Выражения (expressions) (продолжение)

compound_expression::=



Сложное выражение представляет собой комбинацию других выражений и спецсимволов:

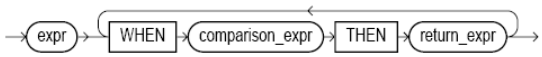
- круглых скобок для изменения порядка вычисления выражений;
- знаков унарных и бинарных арифметических операций;
- знака операции конкатенации строк символов («||»).

Оператор *PRIOR* используется в конструкции *CONNECT BY* иерархических запросов.

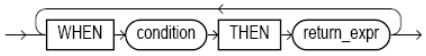
Выражения (expressions) (продолжение)

case_expression::=

simple_case_expression::=



searched_case_expression::=

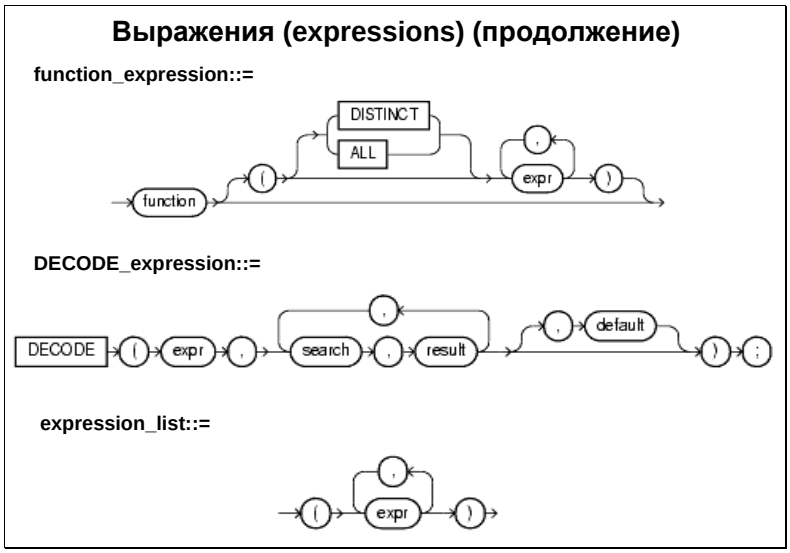


else_clause::=

CASE-выражение позволяет использовать логику условных выражений, не прибегая к услугам традиционного языка программирования.

В случае простого CASE-выражения Oracle ищет первую пару <*comparison_expr*, *return_expr*>, в которой значение *comparison_expr* равно значению проверяемого выражения *expr*, и возвращает значение *return_expr*. Если ни одна из указанных пар не удовлетворяет этому условию, и в выражении используется конструкция *ELSE*, возвращается значение выражения *else_expr*. В противном случае возвращается значение *NULL*.

В случае поискового CASE-выражения Oracle ищет первую пару <*condition*, *return_expr*>, в которой значение условия *condition* равно истине, и возвращает значение *return_expr*. Если ни одна из указанных пар не удовлетворяет этому условию, и в выражении используется конструкция *ELSE*, возвращается значение выражения *else_expr*. В противном случае возвращается значение *NULL*.



В Oracle SQL имеется ряд встроенных функций, которые могут быть вызваны из SQL-операторов. SQL-функции можно разделить на категории в соответствии с типом используемых аргументов.

Кроме того, по числу обрабатываемых строк SQL-функции можно разделить на групповые и однострочные. Групповые функции обрабатывают несколько строк данных и возвращают один результат. Их разрешается применять только в списках выбора и в конструкциях *HAVING* запросов. Однострочные функции обрабатывают одно значение и возвращают другое. Их можно использовать в SQL-операторах, где разрешается применять выражения. Функции этого типа делятся на следующие категории:

- системные переменные,
- числовые функции,
- символьные (текстовые) функции,
- функции для работы с датами,
- функции преобразования данных,
- прочие функции.

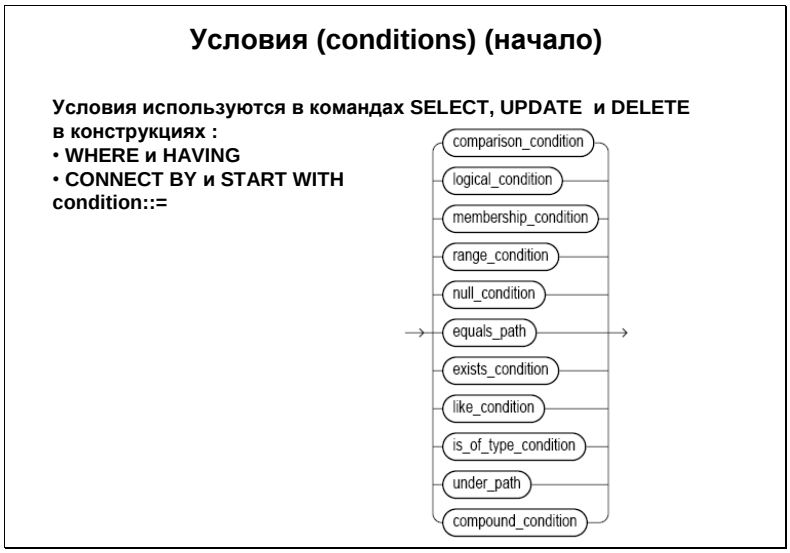
DECODE_expression представляет собой ранее часто используемую конструкцию, которую в последних версиях Oracle заменила конструкция простого CASE-выражения (*simple_case_expression*).

Список выражений (*expression list*) является комбинацией других выражений. Он может быть указан в предикатах сравнения, *in*-предикатах и в предложении *GROUP BY* запросов и подзапросов.

Список выражений может содержать одно или более выражений, разделенных запятыми, а также один или более наборов выражений. Каждый набор выражений в свою очередь может содержать одно или более выражений, разделенных запятыми. В последнем случае (несколько наборов выражений):

- каждый набор выражений заключается в круглые скобки;
- каждый набор выражений должен содержать одно и то же число выражений;
- число выражений в каждом наборе должно соответствовать числу выражений в левой части предиката сравнения или *in*-предиката.

Список выражений в виде выражений, разделенных запятыми, может содержать не более 1000 выражений. Список выражений в виде множества наборов может содержать любое число наборов, но в каждом наборе может быть не более 1000 выражений.



Условие (англ. condition) – комбинация одного или нескольких выражений и логических операторов, определяющая некоторое логическое выражение, которое возвращает значения *true*, *false* или *unknown*. Условия используются в командах *SELECT*, *UPDATE* и *DELETE* в конструкциях:

- *WHERE* и *HAVING*;
- *CONNECT BY* и *START WITH*.

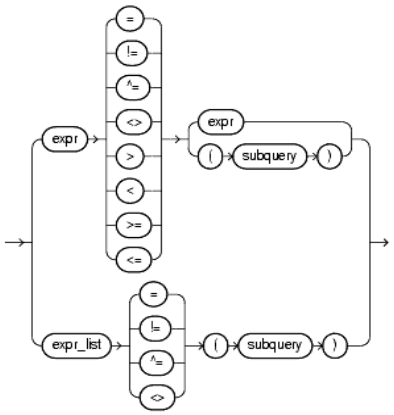
Вычисление логического выражения осуществляется с учетом приоритета операций. Он убывает в соответствии со следующим списком:

- арифметические, строковые и прочие операции;
- операции (предикаты) сравнения;
- прочие предикаты;
- логическая операция *NOT*;
- логическая операция *AND*;
- логическая операция *OR*.

На слайде перечислены все виды грамматических конструкций условий. Мы рассмотрим из них лишь наиболее простые и часто используемые.

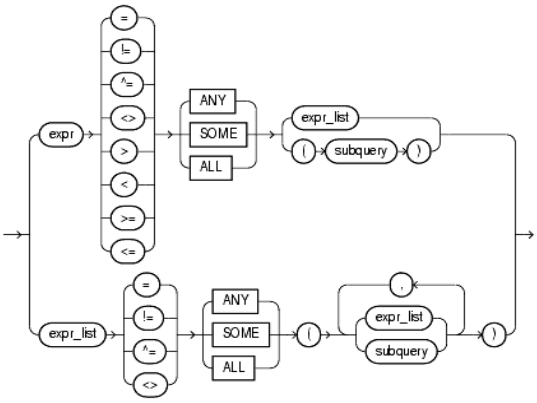
Условия (conditions) (продолжение)

simple_comparison_condition::=



Условия (conditions) (продолжение)

group_comparison_condition::=

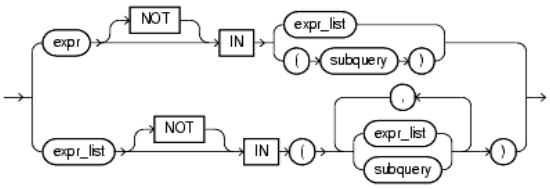


Обратите внимание на то, что в диалекте Oracle в отличие от стандарта SQL/89, предикаты сравнения, предикаты с квантором и *IN*-предикаты допускают сравнение не только скалярных значений (верхние маршруты грамматических диаграмм), но и векторов – списков значений (нижние маршруты диаграмм).

Знаки «!=», «^=», «<>» представляют собой допустимые в SQL-диалекте Oracle знаки операции сравнения «не равно».

Условия (conditions) (продолжение)

membership_condition::=

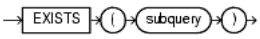


range_condition::=

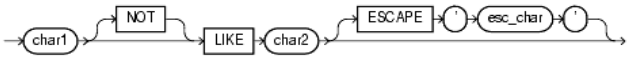
null_condition::=

Условия (conditions) (продолжение)

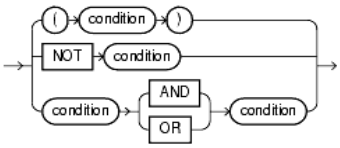
exists_condition::=

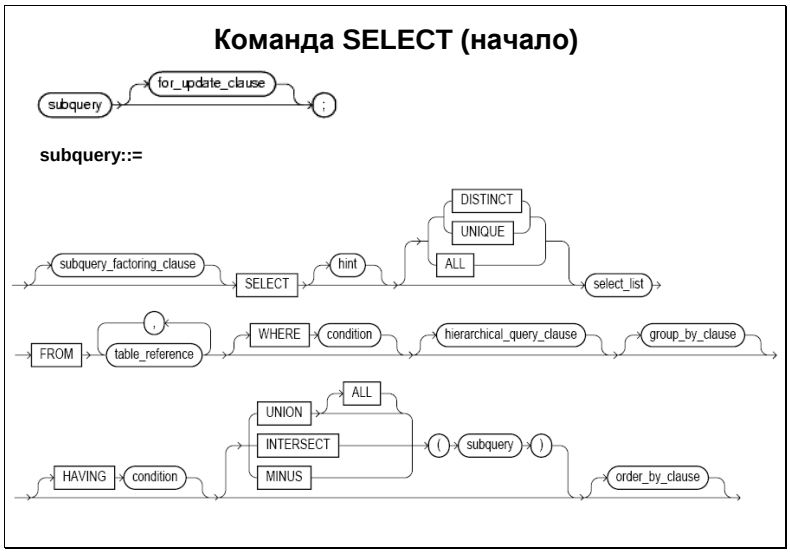


like_condition::=



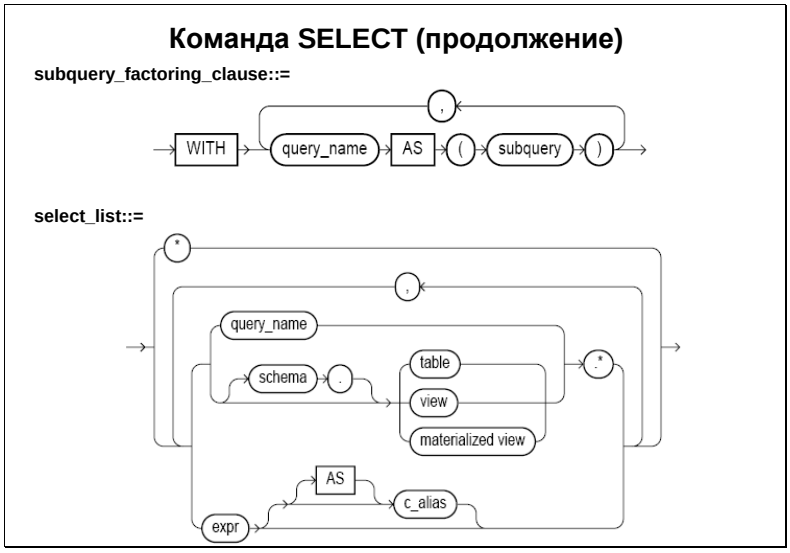
compound_condition::=





Начиная с этого слайда, мы рассматриваем грамматику запросов (подзапросов) на выборку данных диалекта SQL Oracle. Главные конструкции команды *SELECT* сохранили в основном синтаксис и семантику стандарта SQL. Как и ранее нетерминальные символы правых частей будут либо раскрываться далее в соответствующих диаграммах, либо комментироваться в тексте после слайда. Конструкции, относящиеся к объектным расширениям диалекта или к элементам физической модели, оставим без комментариев.

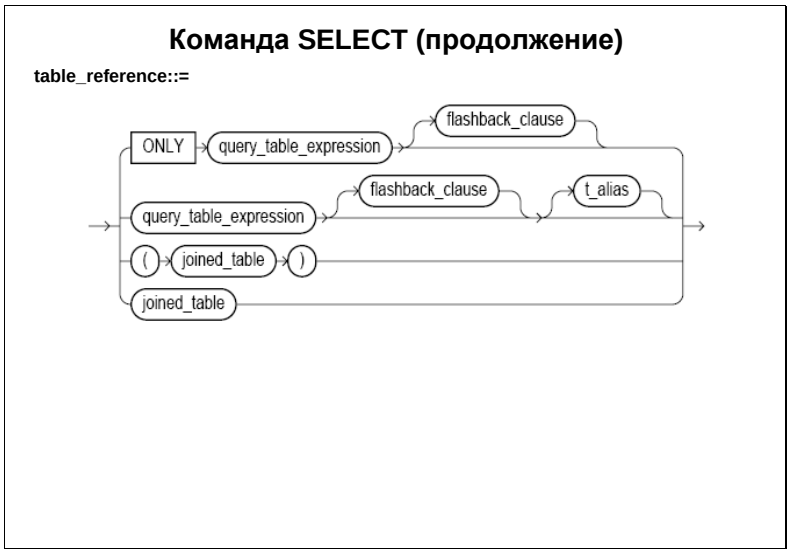
hint определяет в виде комментария инструкции оптимизатору запросов для правильного выбора плана выполнения команды.



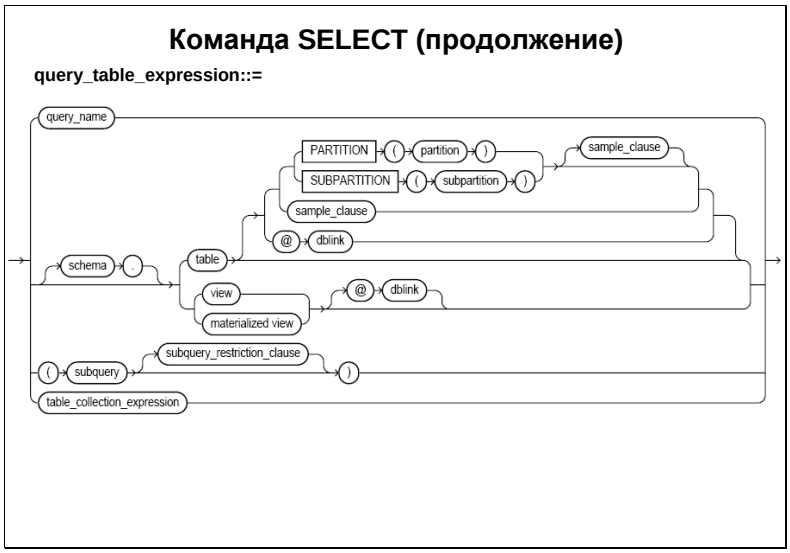
subquery_factoring_clause позволяет присвоить имя (*query_name*) подзапросу (*subquery*) и в дальнейшем многократно использовать его в запросе, указывая, где надо, его имя.

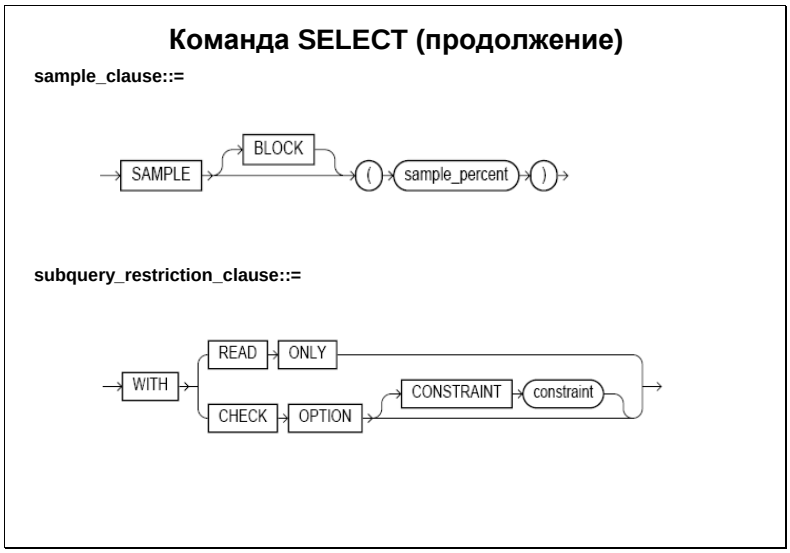
c_alias задает имя столбца в результирующей таблице. Внутри запроса, в котором он введен, его можно использовать только в конструкции *ORDER BY*.

Если весь список выборки (*select_list*) состоит из одного символа «*» (самый верхний маршрут грамматической диаграммы) результирующая таблица будет включать все столбцы всех таблиц запроса. Если символ «*» стоит в контексте одной таблицы (средний маршрут диаграммы) в результирующую таблицу попадут все столбцы указанной таблицы.



t_alias задает синоним табличному выражению (*query_table_expression*), который может использоваться в любом месте внутри запроса, в котором он введен. Он часто необходим для подзапросов, указанных в конструкции *FROM*, а также в случаях, когда одна и та же таблица неоднократно участвует в этой конструкции, или в случаях вложенных подзапросов.



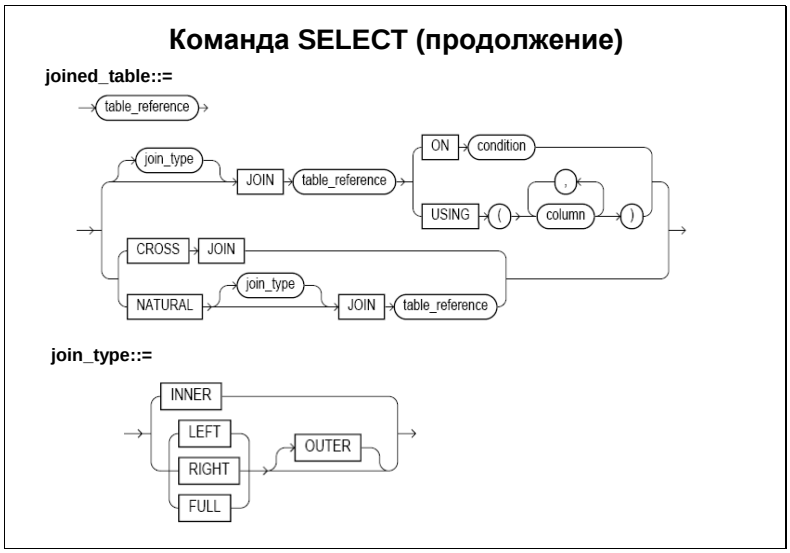


sample_clause указывает Oracle получить результат, обращаясь случайно к части (*sample_percent*) записей (блоков) вместо сканирования всей таблицы.

subquery_restriction_clause используется в командах изменения БД с подзапросом, указывается в подзапросе и позволяет ограничить результат выполнения команды одним из следующих способов:

- *WITH READ ONLY* указывает, что таблица или представление не будут изменяться;
- *WITH CHECK OPTION* указывает, чтобы Oracle запрещал всякие изменения таблицы или представления, которые приведут к появлению строк, не удовлетворяющих подзапросу.

WITH CHECK OPTION CONSTRAINT позволяет задать имя «*WITH CHECK OPTION*»-ограничения (в противном случае ему будет присвоено системное имя).



joined_table используется для задания действия соединения таблиц вместо стандартного Декартова произведения. Oracle рекомендует использовать новый синтаксис с явным указанием типа соединения в конструкции *FROM* вместо старого, при котором условия соединения задавались в конструкции *WHERE*.

join_type определяет тип используемого соединения:

- *INNER* – внутреннее (или естественное) соединение,
- *RIGHT* – правое внешнее соединение,
- *LEFT* – левое внешнее соединение,
- *FULL* – полное внешнее соединение.

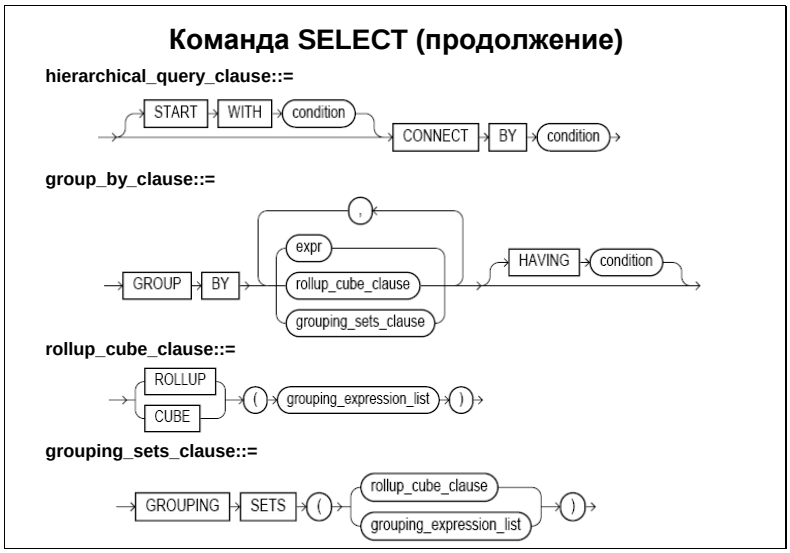
Для трех последних типов соединений можно указать необязательное ключевое слово *OUTER*, что дополнительно будет говорить о том, что соединение является внешним.

В конструкции *ON* можно указать условие соединения любой сложности.

USING может использоваться для эквисоединения по столбцам, которые имеют одинаковые имена в обеих таблицах. Именно их список без дублирования имен указывается после этого ключевого слова. При этом не следует уточнять имя таблицы или ее алиас. Запрос на внешнее соединение с *USING* возвращает по одному экземпляру столбцов соединения, которые объединяют значения столбцов таблиц-аргументов по следующему правилу: если значение из левой таблицы не пусто, берется оно, в противном случае берется значение из другой таблицы.

Ключевые слова *CROSS JOIN* явно указывают на то, что необходима операция Декартова произведения таблиц.

NATURAL указывает на то, что эквисоединение выполняется по всем одноименным столбцам таблиц.



hierarchical_query_clause применяется для выдачи строк в иерархическом порядке (следом за родительской строкой следуют ее дочерние строки). Для этого в таблице с помощью внешнего ключа должна быть представлена иерархическая связь между строками.

START WITH определяет условие, при истинности которого строка считается корневой в иерархии. Таких строк может быть несколько.

CONNECT BY определяет условие, при истинности которого устанавливается связь между родительской и дочерними строками в иерархии. Для указания на родительскую строку в условии перед выражениями со столбцами этой строки задается оператор *PRIOR*.

Иерархический запрос (запрос с конструкцией *hierarchical_query_clause*) может использовать в своем целевом списке псевдостолбец *LEVEL*, возвращающий 1 для корневых строк, 2 для их потомков и т.д.

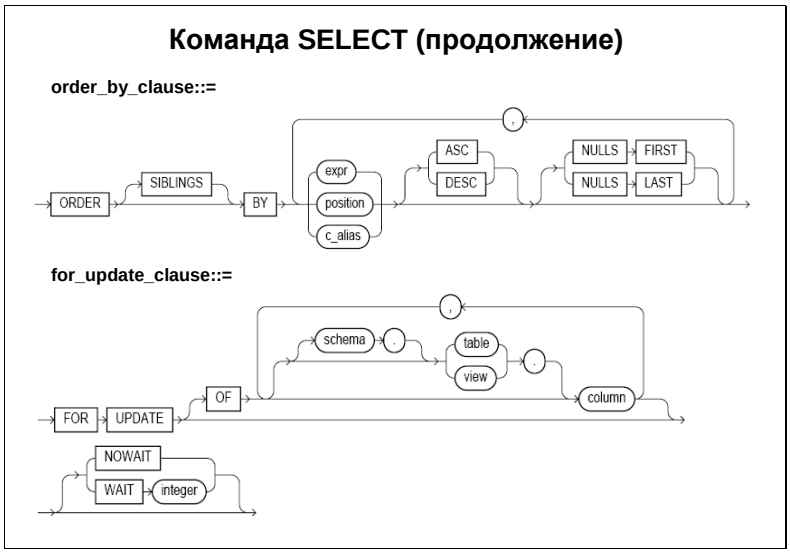
group_by_clause позволяет собирать строки таблиц в группы и строить по одной строке результата для каждой такой группы на основе агрегированной информации о ней. Если в *group_by_clause* указаны конструкции *CUBE* или *ROLLUP*, Oracle производит суперагрегирующее группирование над обычным группированием.

ROLLUP позволяет иметь в результате наряду со строками для групп еще и суперагрегированные строки для супергрупп групп, образующихся при совпадении значений части столбцов группирования.

CUBE позволяет построить группы для любых комбинаций значений указанных в этой конструкции выражений плюс супергруппы для каждого значения и для всей таблицы в целом.

GROUPING SETS позволяет в одной команде использовать несколько принципов группирования одновременно.

Обратите внимание, что конструкция *HAVING*, задающая условие фильтрации групп, может быть указана как совместно с конструкцией *GROUP BY* (нетерминал *group_by_clause* текущего слайда), так и без нее (нетерминал *subquery* первого слайда грамматики команды *SELECT*).



order_by_clause позволяет задать порядок сортировки строк результата команды *SELECT*. Указав через запятую несколько выражений, можно получить иерархическую сортировку – главной является сортировка по первому выражению, группа записей с одинаковыми значениями этого выражения упорядочивается по второму выражению и т.д.

SIBLINGS допустим только в иерархических запросах. Он предотвращает нарушение иерархического порядка строк. Указанное в таком *ORDER BY* упорядочение действует в пределах предков одного потомка.

В списке сортировки могут быть использованы:

- *expr* – допустимое выражение над столбцами таблиц *FROM*,
- *position* – позиция выражения целевого списка (*select_list*) команды *SELECT*,
- *c_alias* – имя столбца результирующей таблицы.

for_update_clause позволяет блокировать селектируемые строки, таким образом защищая их от блокирования и модификации другим пользователем. Эта конструкция может указываться только для самого высокоуровневого *SELECT*.

SQL. Примеры запросов.
Демонстрационная БД Oracle

```
CREATE TABLE dept
    (deptno NUMBER(2) CONSTRAINT pk_dept PRIMARY KEY,
    dname VARCHAR2(14),
    loc VARCHAR2(13) );

CREATE TABLE emp
    (empno NUMBER(4) CONSTRAINT pk_emp PRIMARY KEY,
    ename VARCHAR2(10),
    job VARCHAR2(9),
    mgr NUMBER(4) CONSTRAINT fk_empno REFERENCES
                                    emp (empno),
    hiredate DATE,
    sal NUMBER(7,2),
    comm NUMBER(7,2),
    deptno NUMBER(2) CONSTRAINT fk_deptno REFERENCES
                                    dept (deptno));
```

Далее мы рассмотрим ряд примеров запросов на языке SQL, демонстрирующих использование конструкций команды *SELECT*. Первая часть примеров показывает преимущественно работу специфических конструкций диалекта Oracle. Поэтому в качестве БД для них использована стандартная фирменная демонстрационная БД, поставляемая вместе с сервером Oracle уже несколько поколений системы. Она формируется при создании новой пользовательской БД в схеме пользователя с именем *scott* (пароль *tiger*). На слайде приведены команды создания таблиц *dept* (отделы фирмы) и *emp* (служащие этих отделов), определяющие их структуру и ограничения целостности.

В частности, в каждой таблице определены первичные ключи *deptno* и *empno*, соответственно. С помощью внешнего ключа *deptno* таблицы *emp* представлены связи типа 1:*M* между отделами и служащими, а внешний ключ *mgr* таблицы *emp*, ссылающийся на первичный ключ этой же таблицы, задает иерархию подчинения служащих.

SQL. Примеры запросов.
Демонстрационная БД Oracle

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850		30
7782	CLARK	MANAGER	7839	09-JUN-81	2450		10
7788	SCOTT	ANALYST	7566	19-APR-87	3000		20
7839	KING	PRESIDENT		17-NOV-81	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-81	1500	0	30
7876	ADAMS	CLERK	7788	23-MAY-87	1100		20
7900	JAMES	CLERK	7698	03-DEC-81	950		30
7902	FORD	ANALYST	7566	03-DEC-81	3000		20
7934	MILLER	CLERK	7782	23-JAN-82	1300		10

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

На слайде приведено стандартное содержимое (или экстенсионалы) таблиц *emp* и *dept*. Обратите внимание на тот факт, что в отделе с номером 40 никто не работает. Также следует отметить, что президент компании (служащий с номером 7839) не имеет руководителя, а, следовательно, в его строке таблицы *emp* атрибут *mgr* принимает значение *NULL*. Эти особенности сыграют свою роль в некоторых запросах.

SQL. Примеры запросов

```
SELECT * FROM emp WHERE deptno = 30;

SELECT ename, job, sal, deptno FROM emp
      WHERE NOT (job = 'SALESMAN' AND deptno = 30);

SELECT          a.deptno "Department",
                a.num_emp * 100 / b.total_count "%Employees",
                a.sal_sum * 100 /b.total_sal "%Salary"
      FROM (SELECT deptno, COUNT(*) num_emp, SUM(sal) sal_sum
                FROM scott.emp GROUP BY deptno) a,
            (SELECT COUNT(*) total_count, SUM(sal) total_sal
                FROM scott.emp) b ;

SELECT COUNT(*) * 10 FROM emp SAMPLE BLOCK (10);
```

Запросы в первых двух примерах осуществляют простые выборки из таблицы *emp* и, по-видимому, в пояснениях не нуждаются.

Третий пример демонстрирует запрос, использующий подзапросы в конструкции *FROM*. Разбор такой команды лучше начинать с этих подзапросов. Первый из них (обозначенный алиасом «*a*») группирует строки таблицы *emp* по номерам отделов (*depno*). Результирующая таблица этого подзапроса будет иметь 3 строки (у нас есть служащие только в трех отделах) со значениями номера отдела (*deptno*), количества служащих в нем (*COUNT*(*)) и их суммарного дохода (*SUM*(*sal*)). Благодаря используемым в целевом списке алиасам столбцов, второй и третий столбцы этой таблицы будут иметь имена *num_emp* и *sal_sum* соответственно. Именно по этим именам мы будем обращаться к ним во внешней команде *SELECT*.

Второй подзапрос (обозначенный алиасом «*b*»), хоть и не содержит *GROUP BY*, также использует группировку, благодаря указанным в целевом списке функциям агрегирования. Это неявное группирование порождает в результирующей таблице подзапроса одну строку со значениями общего количества служащих в фирме (*COUNT*(*)) и их суммарного дохода (*SUM*(*sal*)). Алиасы столбцов (*total_count* и *total_sal*) задают их имена.

Таким образом, в списке *FROM* указаны лишь два подзапроса, намеков на то или иное их соединение нет, значит, система выполнит их Декартово произведение и к каждой строке таблицы *a* добавит строку таблицы *b*. Это позволит породить результат всего запроса в виде таблицы с тремя строками (по одной для каждого отдела), в столбцах которой будут содержаться: номер отдела (столбец с именем *Department*), относительное число служащих (%*Employees*) и их относительный суммарный доход (%*Salary*).

Четвертый запрос показывает, как быстро (в данном случае в 10 раз быстрее сканирования всей таблицы) можно получить оценку числа служащих в фирме, обратившись случайно к части (в данном случае к 10%) блоков таблицы.

SQL. Примеры запросов

```
SELECT deptno, MIN(sal), MAX (sal) FROM emp GROUP BY deptno;
DEPTNO MIN(SAL) MAX(SAL)
10      1300      5000
20      800       3000
30      950       2850

SELECT deptno, MIN(sal), MAX (sal) FROM emp
                WHERE job = 'CLERK' GROUP BY deptno;
DEPTNO MIN(SAL) MAX(SAL)
10      1300      1300
20      800       1100
30      950       950

SELECT deptno, MIN(sal), MAX (sal) FROM emp
                WHERE job = 'CLERK' GROUP BY deptno HAVING MIN(sal) < 1000;
DEPTNO MIN(SAL) MAX(SAL)
20      800       1100
30      950       950
```

Запросы этого слайда демонстрируют варианты явной группировки строк таблицы *emp* по номеру отдела (*deptno*): простая группировка, группировка с предварительной фильтрацией строк и группировка с фильтрацией групп.

SQL. Примеры запросов

```
SELECT LPAD(' ', 2 * (LEVEL - 1)) || ename org_chart, empno, mgr, job
FROM emp START WITH job = 'PRESIDENT'
CONNECT BY PRIOR empno = mgr;
```

ORG_CHART	EMPNO	MGR	JOB
KING	7839		PRESIDENT
JONES	7566	7839	MANAGER
SCOTT	7788	7566	ANALYST
ADAMS	7876	7788	CLERK
FORD	7902	7566	ANALYST
SMITH	7369	7902	CLERK
BLAKE	7698	7839	MANAGER
ALLEN	7499	7698	SALESMAN
WARD	7521	7698	SALESMAN
MARTIN	7654	7698	SALESMAN
TURNER	7844	7698	SALESMAN
JAMES	7900	7698	CLERK
CLARK	7782	7839	MANAGER
MILLER	7934	7782	CLERK

На слайде представлен иерархический запрос (с характерными конструкциями *START WITH* и *CONNECT BY*), обеспечивающий выдачу строк таблицы *emp* в порядке подчинения служащих. Иерархия служащих обходится по принципу «сначала вглубь – затем вширь».

Условие поиска корней дерева указано после ключевых слов *START WITH*, в данном случае выбираются служащие с должностью *PRESIDENT*. В демонстрационной БД такой служащий один – по фамилии *KING*. В конструкции *CONNECT BY* указан критерий для поиска ближайших детей вершины дерева. Родительским атрибутам в нем сопутствует ключевое слово *PRIOR*. В нашем примере дочерними строками будут являться те строки таблицы *emp*, в которых значение атрибута *mgr* равно значению первичного ключа (*empno*) родительской записи.

Мы уже упоминали, что в иерархических запросах допустимо использование псевдостолбца *LEVEL*, значение которого определяет уровень строки в иерархии (для корневых строк он равен 1, для детей корневых строк – 2 и т.д.). Применение *LEVEL* в функции *LPAD* первого столбца результата запроса обеспечивает наглядное представление иерархии строк за счет добавления слева от значения фамилии служащего по два пробела для каждого уровня дерева, начиная со второго.

SQL. Примеры запросов

```
SELECT DECODE(GROUPING(dname), 1, 'All Departments', dname) AS dname,
       DECODE(GROUPING(job), 1, 'All Jobs', job) AS job,
       COUNT(*) "Total Empl", AVG(sal) * 12 "Average Sal"
FROM emp JOIN dept USING (deptno) GROUP BY CUBE (dname, job);
```

DNAME	JOB	Total Empl	Average Sal
All Departments	All Jobs	14	24878,57143
All Departments	CLERK	4	12450
All Departments	ANALYST	2	36000
All Departments	MANAGER	3	33100
All Departments	SALESMAN	4	16800
All Departments	PRESIDENT	1	60000
SALES	All Jobs	6	18800
SALES	CLERK	1	11400
SALES	MANAGER	1	34200
SALES	SALESMAN	4	16800
RESEARCH	All Jobs	5	26100
RESEARCH	CLERK	2	11400
RESEARCH	ANALYST	2	36000
RESEARCH	MANAGER	1	35700
ACCOUNTING	All Jobs	3	35000
ACCOUNTING	CLERK	1	15600
ACCOUNTING	MANAGER	1	29400
ACCOUNTING	PRESIDENT	1	60000

Этот запрос демонстрирует технику «кубической» группировки (конструкция *GROUP BY CUBE*). Она напоминает принципы работы с многомерными кубами в OLAP-технологии. Но обо всем по порядку.

Исходными для построения куба будут являться строки таблицы, построенной естественным соединением таблиц *emr* и *dept* по столбцам *deptno* (тот факт, что столбцы связи имеют одинаковые имена, позволяет нам использовать конструкцию *USING*). Указанные в списке группировки столбцы *dname* и *job*, задают в данном случае две оси системы координат куба. Точки оси соответствуют различным значениям соответствующего столбца (*SALES, RESEARCH, ACCOUNTING* – для столбца *dname* и *CLERK, ANALYST, MANAGER, SALESMAN, PRESIDENT* – для столбца *job*). Автоматически на каждую ось добавляется по одной точке, олицетворяющей все различные значения столбца.

Точка куба (в данном случае ее можно мыслить как точку в двумерном пространстве с ортогональными осями координат) представляет собой группу строк исходной таблицы со значениями столбцов, соответствующими точкам каждой оси. Именно для этих групп применяются функции агрегирования, указанные в целевом списке команды *SELECT* для столбцов *Total Empl* и *Average Sal* и вычисляющие количество служащих в группе и их средний годовой доход соответственно.

Первые два столбца результата обеспечивают идентификацию групп. Для этого используются встроенные функции *DECODE* и *GROUPING*. Функция *GROUPING* применяется к выражению группировки (в нашем случае выражения группировки представляют собой имена столбцов) и принимает значение 1, если точка на этой оси для группы соответствует множеству значений, и значение 0, если одному конкретному значению. Функция *DECODE* для первого столбца результата работает следующим образом: если текущая группа соответствует множеству значений по оси столбца *dname*, в результирующую строку для этой группы помещаем значение *All Departments*, если текущая группа соответствует одному значению по оси этого столбца, в результирующую строку для группы помещаем это значение. Работа этой функции для второго столбца аналогична.

SQL. Примеры запросов

```
SELECT ename, deptno
     FROM emp WHERE deptno =
       (SELECT deptno FROM emp WHERE ename = 'TURNER');

SELECT ename, job, deptno, dname
       FROM emp NATURAL JOIN dept
       WHERE job = 'CLERK';
```

ENAME	JOB	DEPTNO	DNAME
SMITH	CLERK	20	RESEARCH
ADAMS	CLERK	20	RESEARCH
JAMES	CLERK	30	SALES
MILLER	CLERK	10	ACCOUNTING

Первый запрос слайда демонстрирует работу предиката сравнения с подзапросом в правой части. Напоминаем, что для него подзапрос должен возвращать не более одной строки. В противном случае будет выдано сообщение об ошибке. Избежать ее даже в случае нескольких строк подзапроса поможет предикат включения *IN*.

Во втором запросе используется конструкция *NATURAL JOIN*, осуществляющая операцию естественного соединения по одноименным столбцам таблиц. С таким же успехом можно было в данном случае воспользоваться конструкциями *JOIN USING* и *JOIN ON*.

SQL. Примеры запросов

```
SELECT e1.ename||' works for '||e2.ename "Employees and their Managers"
      FROM emp e1 JOIN emp e2 ON e1.mgr = e2.empno;

Employees and their Managers
SMITH works for FORD
ALLEN works for BLAKE
WARD works for BLAKE
JONES works for KING
MARTIN works for BLAKE
BLAKE works for KING
CLARK works for KING
SCOTT works for JONES
TURNER works for BLAKE
ADAMS works for SCOTT
JAMES works for BLAKE
FORD works for JONES
MILLER works for CLARK
```

А вот в этом запросе можно использовать лишь последний вариант, а именно, *JOIN ON*, поскольку имена столбцов соединения не совпадают. Кстати, в этом запросе не обойтись и без алиасов таблиц (*e1* и *e2*), поскольку в нем дважды используется одна и та же таблица.

SQL. Примеры запросов

```
SELECT ename, job, dept.deptno, dname
     FROM emp RIGHT OUTER JOIN dept ON
     emp.deptno = dept.deptno AND job = 'CLERK';

ENAME      JOB    DEPTNO    DNAME
SMITH      CLERK      20    RESEARCH
ADAMS      CLERK      20    RESEARCH
JAMES      CLERK      30    SALES
MILLER     CLERK      10    ACCOUNTING
                      40    OPERATIONS

SELECT SYSDATE FROM DUAL;

SELECT zseq.nextval FROM dual;
```

На слайде показан запрос с правым внешним соединением (*RIGHT OUTER JOIN*). Как видим, несмотря на то, что в 40-ом отделе нет ни одного служащего с должностью *CLERK* (да и вообще, там никто не работает), строка о нем из таблицы *dept* была слева сконкатенирована с пустой строкой и добавлена в результирующую таблицу.

Два последних примера из группы запросов, демонстрирующих преимущественно особенности диалекта SQL Oracle, показывают использование таблицы *DUAL*.

Таблица с именем *DUAL* создается системой автоматически вместе с новой БД в схеме пользователя *SYS* и под этим именем доступна всем пользователям БД. Схема таблицы содержит всего один столбец с именем *DUMMY*, определенный как *VARCHAR2(1)*. Экстенсионал таблицы состоит из одной записи со значением столбца *DUMMY*, равным «*X*». Часто бывает полезно построить таблицу с такой же структурой, как *DUAL*, но содержащую в своем теле значение константы, системной функции (например, *SYSDATE*), псевдостолбца (например, *nextval*) или более сложного выражения. В таких случаях незаменима таблица *DUAL*.

SQL. Примеры запросов

Получить фамилии хирургов

```
SELECT Фамилия FROM ВРАЧ
  WHERE Специальность = 'ХИРУРГ'
```

Получить фамилии врачей, лечащих больных палаты №2 больницы №5

```
SELECT В.Фамилия
  FROM (ВРАЧ В JOIN ВРАЧ-ПАЦИЕНТ ВП USING (К/В))
  JOIN РАЗМЕЩЕНИЕ Р USING (Р/Н)
  WHERE Р. К/Б = 5 AND Р. Н/П = 2

SELECT Фамилия FROM ВРАЧ WHERE К/В IN
  (SELECT К/В FROM ВРАЧ-ПАЦИЕНТ WHERE Р/Н IN
        (SELECT Р/Н FROM РАЗМЕЩЕНИЕ
              WHERE К/Б = 5 AND Н/П =2))
```

Этим слайдом мы начинаем серию примеров «семантически значимых» запросов на выборку данных из нашей демонстрационной медицинской БД.

Первые запросы относительно просты и в пояснениях не нуждаются. Следует отметить лишь, что часто один и тот же запрос на языке SQL может быть выражен несколькими различными способами, как в случае второго запроса.

SQL. Примеры запросов

Получить фамилии врачей, лечащих всех больных палаты №2 больницы №5

```
SELECT Фамилия FROM ВРАЧ WHERE К/В IN
  (SELECT К/В FROM ВРАЧ-ПАЦИЕНТ WHERE Р/Н IN
        (SELECT Р/Н FROM РАЗМЕЩЕНИЕ
        WHERE К/Б = 5 AND Н/П =2)
        GROUP BY К/В
        HAVING COUNT(*) = (SELECT COUNT(*) FROM
        РАЗМЕЩЕНИЕ WHERE К/Б = 5 AND Н/П =2))
```

Получить полный список специальностей врачей

```
SELECT DISTINCT Специальность FROM ВРАЧ
```

Получить названия больниц, которые имеют более 5 педиатров

```
SELECT Название FROM БОЛЬНИЦА WHERE К/Б IN
  (SELECT К/Б FROM ВРАЧ
  WHERE Специальность = 'ПЕДИАТР'
  GROUP BY К/Б
  HAVING COUNT(*) > 5)
```

В первом запросе этого слайда средний подзапрос возвращает *К/В* тех врачей, для которых количество его пациентов, лежащих в указанной палате, совпадает с числом всех пациентов этой палаты.

SQL. Примеры запросов

```
Получить фамилии врачей, не лечащих пациента с P/H = 111111
    SELECT Фамилия FROM ВРАЧ B WHERE NOT EXISTS
      (SELECT K/B FROM ВРАЧ-ПАЦИЕНТ WHERE
      K/B = B.K/B AND P/H = 111111)

Получить фамилии врачей, лечащих всех пациентов
    SELECT Фамилия FROM ВРАЧ WHERE K/B IN
      (SELECT K/B FROM ВРАЧ-ПАЦИЕНТ
      GROUP BY K/B
      HAVING COUNT(*) =
            (SELECT COUNT(*) FROM ПАЦИЕНТ))

Получить фамилии врачей, лечащих по крайней мере одного пациента
    врача с кодом 999
    SELECT Фамилия FROM ВРАЧ WHERE K/B IN
      (SELECT K/B FROM ВРАЧ-ПАЦИЕНТ WHERE P/H IN
            (SELECT P/H FROM ВРАЧ-ПАЦИЕНТ
                  WHERE K/B =999))
```

Внутренний подзапрос в первом примере является коррелированным подзапросом (англ. correlated subquery), поскольку в его условии используется значение столбца *К/В* из текущей строки внешнего запроса (о чем говорит алиас *В*, указанный для этого столбца и введенный во внешнем запросе для таблицы *ВРАЧ*). Для подзапросов, используемых в предикате *EXISTS*, целевой список не играет принципиальной роли (с таким же успехом вместо *К/В* в нем могла стоять «*»).

Внутренний подзапрос второго примера находит значения *К/В* тех врачей, у которых количество его пациентов совпадает с общим числом пациентов, хранящихся в таблице *ПАЦИЕНТ*. Такая реализация этого запроса возможна благодаря тому, что группа столбцов {*К/В*, *Р/Н*} объявлена в таблице *ВРАЧ-ПАЦИЕНТ* как первичный ключ, а значит, система не позволит ввести строки-дубликаты.

Последний запрос этого слайда открывает группу примеров, посвященных сравнению множеств пациентов врачей. Этот запрос представляет собой типичный запрос с квантором существования – врач будет возвращаться в результате, если пересечение его множества пациентов с множеством пациентов другого врача (в примере – с кодом *999*) не пусто.

SQL. Примеры запросов

```
Получить фамилии врачей, лечащих всех пациентов врача с кодом 999
  и может быть еще кого-то
  SELECT Фамилия FROM ВРАЧ WHERE К/В IN
        (SELECT К/В
        FROM ВРАЧ-ПАЦИЕНТ ВП
        LEFT OUTER JOIN ВРАЧ-ПАЦИЕНТ ВП999
        ON ВП.Р/Н = ВП999.Р/Н AND ВП999.К/В = 999
          GROUP BY ВП.К/В
          HAVING COUNT (DISTINCT ВП999.Р/Н) =
            (SELECT COUNT(*) FROM ВРАЧ-ПАЦИЕНТ  WHERE К/В = 999))
Получить фамилии врачей, лечащих всех пациентов врача с кодом 999
  и обязательно еще кого-то
HAVING COUNT (DISTINCT ВП999.Р/Н) =
            (SELECT COUNT(*) FROM ВРАЧ-ПАЦИЕНТ WHERE К/В = 999)
          AND COUNT (*) >
            (SELECT COUNT(*) FROM ВРАЧ-ПАЦИЕНТ WHERE К/В = 999)
Получить фамилии врачей, лечащих не всех пациентов врача с кодом
  999
HAVING COUNT (DISTINCT ВП999.Р/Н) <>
            (SELECT COUNT(*) FROM ВРАЧ-ПАЦИЕНТ WHERE К/В = 999)
```

Другие запросы этой группы имеют скрытый квантор всеобщности.

Первый запрос этого слайда проверяет множества пациентов на включение. Действительно, он получит врачей, для которых его множество пациентов включает (строго или нестрого) множество пациентов врача с кодом *999*.

Для второго и третьего запросов для экономии места на слайде приведены только конструкции *HAVING*. Все остальные конструкции этих запросов буквально совпадают с конструкциями первого запроса.

Второй запрос проверяет множества пациентов на строгое включение. Он получит врачей, для которых его множество пациентов строго включает множество пациентов врача с кодом *999*.

Третий запрос проверяет множества пациентов врачей на отсутствие включения.

Поясним работу приведенных на слайде запросов.

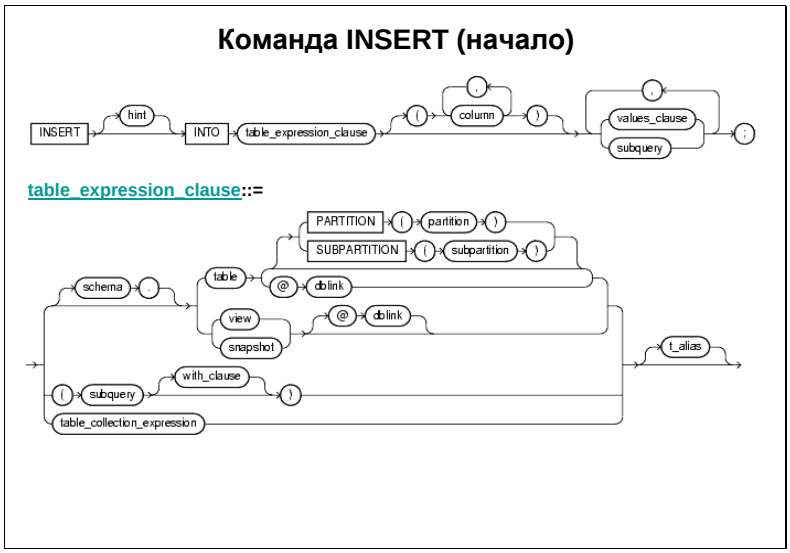
В результате левого внешнего соединения к каждому кортежу отношения *ВРАЧ-ПАЦИЕНТ* (алиас *ВП*) добавится или нет кортеж того же отношения, относящийся к тому же пациенту, но говорящий о его связи с врачом, имеющим код *999* (алиас *ВП999*). Конструкция *GROUP BY* соберет в одну группу все связи с пациентами одного врача.

Выражение *COUNT* (*DISTINCT ВП999.Р/Н*) посчитает количество общих пациентов у врача группы и врача с кодом *999*, поскольку для «необщих» пациентов значение атрибута *ВП999.Р/Н* равно *NULL* и функцией *COUNT* в расчет не принимается.

В первом запросе оно сравнивается с количеством пациентов врача с кодом *999*. Эта проверка означает, что мощность пересечения множеств равна мощности второго множества, что гарантирует наличие отношения включения.

Во втором запросе кроме этого требуется еще, чтобы всего пациентов у врача группы было больше чем у врача с кодом *999*. Что обеспечивает наличие строгого включения.

В третьем запросе достаточно проверить, что мощность пересечения множеств не равна мощности второго множества.



Команда *INSERT* используется для добавления строк в таблицы и представления.

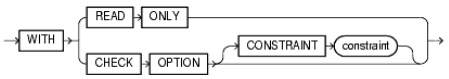
table_expression_clause определяет объект БД (обычно, таблицу), куда добавляются записи. Вместо прямого указания на объект БД можно использовать подзапрос к нему. Это обеспечивает дополнительную функциональность.

column определяет столбец таблицы или представления, для которого в команде будут предусмотрены значения в создаваемых строках. Если список столбцов опущен, предполагается, что будут заданы значения для всех столбцов, причем в порядке, совпадающем с порядком столбцов команды *CREATE TABLE*.

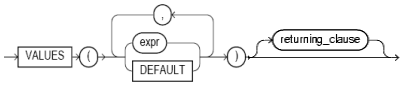
subquery как альтернатива *values_clause* дает возможность вставлять строки (несколько за одну команду), которые являются результатом подзапроса. Использование *values_clause* позволяет в одной команде создать лишь одну строку. Столбцы результата подзапроса должны соответствовать столбцам команды *INSERT* (указанным явно или подразумеваемым).

Команда INSERT (продолжение)

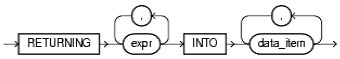
with_clause::=



values_clause::=



returning_clause::=



with_clause используется в командах *INSERT* с подзапросом в качестве *table_expression_clause*, указывается в подзапросе и позволяет ограничить результат выполнения команды одним из следующих способов:

- *WITH READ ONLY* указывает, что таблица или представление не будут изменяться;
- *WITH CHECK OPTION* указывает, чтобы Oracle запрещал всякие изменения таблицы или представления, которые приведут к появлению строк, не удовлетворяющих подзапросу.

WITH CHECK OPTION CONSTRAINT позволяет задать имя «*WITH CHECK OPTION*»-ограничения (в противном случае ему будет присвоено системное имя).

values_clause определяет вводимую строку, которая задается литерально в виде списка выражений.

returning_clause позволяет возвратить в вызвавшую команду программу значения выражений, включающих значения столбцов вновь созданной строки. Список этих выражений указывается после слова *RETURNING*, а следом после слова *INTO* задаются имена переменных, объявленных в программе. Количество и типы выражений и переменных должны соответствовать друг другу.

SQL. Примеры запросов. INSERT

```
INSERT INTO dept
      VALUES (50, 'PRODUCTION', 'SAN FRANCISCO');
INSERT INTO emp (empno, ename, job, sal, comm, deptno)    VALUES (7890,
      'JINKS', 'CLERK', 1.2E3, NULL, 40);
INSERT INTO (SELECT empno, ename, job, sal, comm, deptno FROM emp)
      VALUES (7890, 'JINKS', 'CLERK', 1.2E3, NULL, 40);
INSERT INTO bonus
      SELECT ename, job, sal, comm    FROM emp    WHERE comm > 0.25 *
      sal    OR job IN ('PRESIDENT', 'MANAGER');
INSERT INTO emp    VALUES  (empseq.nextval, 'LEWIS', 'CLERK', 7903,
      SYSDATE, 1200, NULL, 20);
INSERT INTO emp VALUES (empseq.nextval, 'LEWIS', 'CLERK', 7903,
      SYSDATE, 1200, NULL, 20)    RETURNING sal*12, job INTO :bnd1,
      :bnd2;
INSERT INTO (SELECT empno, ename, deptno FROM emp
      WHERE deptno < 10)    VALUES (8903, 'Taylor', 20);
INSERT INTO (SELECT empno, ename, deptno FROM emp
      WHERE deptno < 10 WITH CHECK OPTION) VALUES (8903, 'Taylor', 20);
```

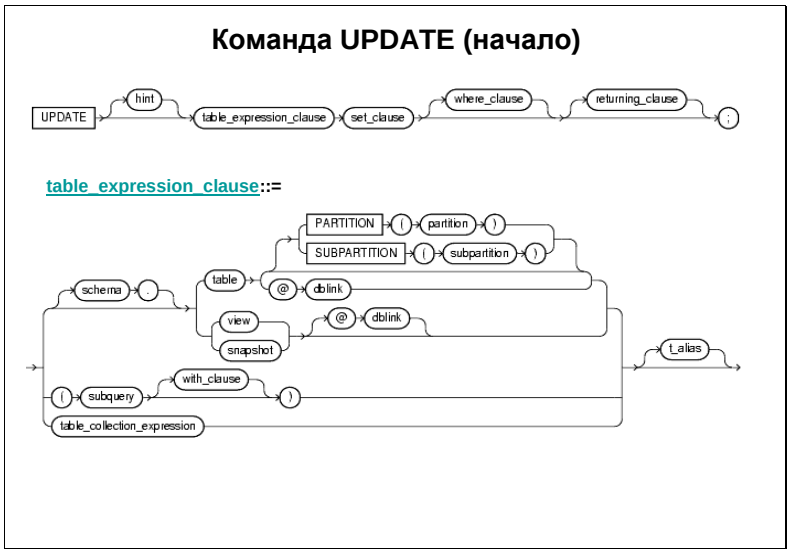
Продемонстрируем ряд примеров команды *INSERT*.

Первые три команды безусловно добавляют в указанные таблицы по одному кортежу, заданному литерально (константами). В первом случае не используется список имен столбцов, и поэтому кортеж задан полностью, причем порядок значений соответствует порядку столбцов в таблице. Второй и третий запросы эквивалентны и демонстрируют два подхода, применимые в случае, когда новый кортеж задается частично. При первом используется список столбцов, при втором – подзапрос.

Четвертый пример добавляет в таблицу *bonus* результат обработки подзапроса. Схема таблицы в точности соответствует целевому списку подзапроса.

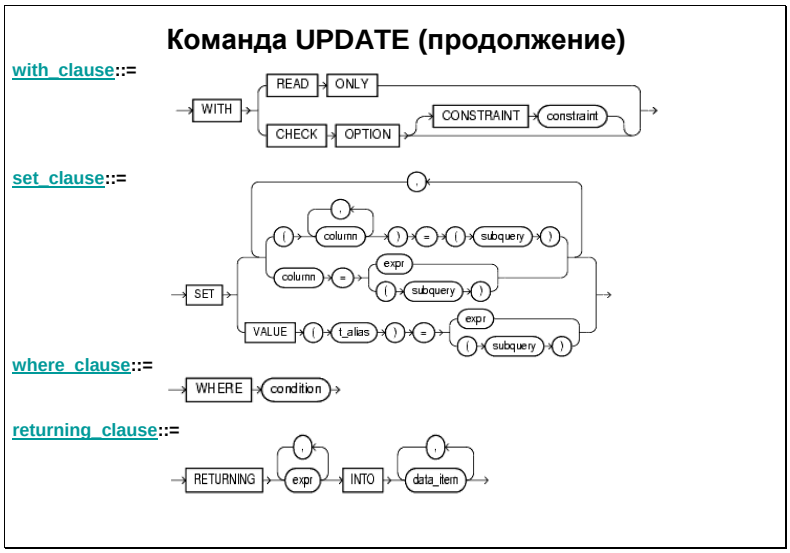
Пятый и шестой примеры демонстрируют обращения к последовательности с именем *empseq* за очередным значением (*nextval*) для суррогатного ключа (в данном случае – столбец *empno*) и к функции *SYSDATE* за системной датой и временем. Шестой пример кроме этого пишет в программные переменные *bnd1* и *bnd2* годовой доход и должность нового сотрудника.

Две последних команды отличаются лишь конструкцией *WITH CHECK OPTION*, стоящей в подзапросе второй из них. Благодаря ей включение нового кортежа в таблицу *emp* во втором случае не произойдет, так как добавляемый кортеж не удовлетворяет условию подзапроса.



Команда *UPDATE* предназначена для изменения значений в строках таблиц и представлений.

table_expression_clause определяет объект БД (обычно, таблицу), где модифицируются записи. Вместо прямого указания на объект БД можно использовать подзапрос к нему. Это обеспечивает дополнительную функциональность.



with_clause используется в командах *UPDATE* с подзапросом в качестве *table_expression_clause*, указывается в подзапросе и позволяет ограничить результат выполнения команды одним из следующих способов:

- *WITH READ ONLY* указывает, что таблица или представление не будут изменяться;
- *WITH CHECK OPTION* указывает, чтобы Oracle запрещал всякие изменения таблицы или представления, которые приведут к появлению строк, не удовлетворяющих подзапросу.

WITH CHECK OPTION CONSTRAINT позволяет задать имя «*WITH CHECK OPTION*»-ограничения (в противном случае ему будет присвоено системное имя).

set_clause устанавливает новые значения столбцов, указанных в левых частях операций присвоения. Новые значения определяются выражениями или подзапросами правых частей. В операции присвоения для списка столбцов (верхний маршрут диаграммы) подзапрос должен возвращать одну строку с соответствующими значениями. В операции присвоения для одного столбца (средний маршрут диаграммы) подзапрос должен возвращать одну строку с одним значением. Операция присвоения последнего вида (начинается со слова *VALUE*) используется для объектных таблиц.

where_clause определяет условие фильтрации строк изменяемого объекта БД (таблицы или представления). Указанные в команде изменения произойдут только в строках, для которых это условие даст значение «*истина*». Если условие отсутствует, изменениям подвергнутся все строки объекта.

returning_clause позволяет возвратить в вызвавшую команду программу значения выражений, включающих значения модифицируемых столбцов. Список этих выражений указывается после слова *RETURNING*, а следом после слова *INTO* задаются имена переменных, объявленных в программе. Количество и типы выражений и переменных должны соответствовать друг другу.

SQL. Примеры запросов. UPDATE

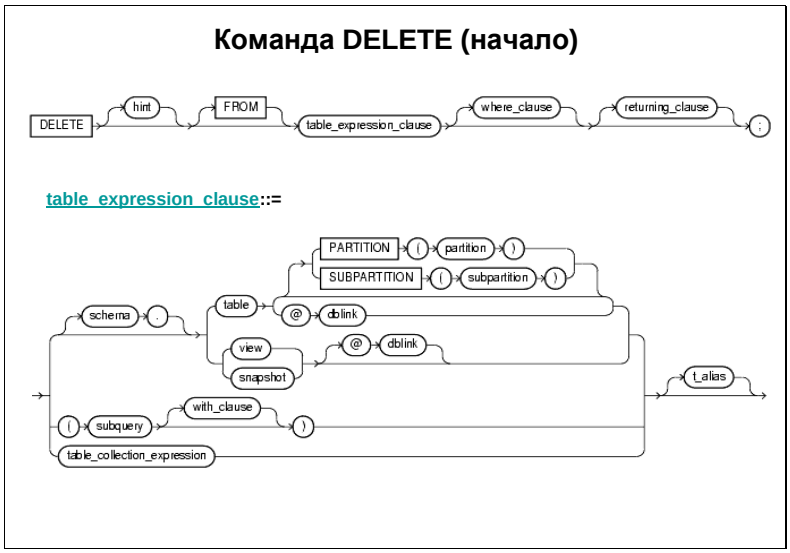
```
UPDATE emp SET comm = NULL WHERE job = 'TRAINEE';
UPDATE emp SET job = 'MANAGER', sal = sal + 1000, deptno =
    20 WHERE ename = 'JONES';
UPDATE emp a
    SET deptno = (SELECT deptno FROM dept WHERE
                  loc = 'BOSTON'),
    (sal, comm) = (SELECT 1.1*AVG(sal), 1.5*AVG(comm)
            FROM emp b WHERE a.deptno = b.deptno)
    WHERE deptno IN (SELECT deptno FROM dept
      WHERE loc = 'DALLAS' OR loc = 'DETROIT');
UPDATE emp  SET job ='MANAGER', sal = sal + 1000, deptno =
    20  WHERE ename = 'JONES‘
    RETURNING sal*0.25, ename, deptno INTO :bnd1, :bnd2,
    :bnd3;
```

На слайде показан ряд примеров команды *UPDATE*.

Первый из них делает неопределенным (*NULL*) значения комиссионных для стажеров (служащих с должностью *TRAINEE*). Второй запрос делает служащего по фамилии *JONES* менеджером 20-го отдела и увеличивает ему доход на 1000.

Область действия третьего запроса – служащие отделов, расположенных в Далласе и Детройте (условие *WHERE*). Они переводятся в отдел, дислоцированный в Бостоне, их доход и комиссионные становятся соответственно на 10% и 50% больше средних доходов и комиссионных служащих их предыдущих отделов.

Последний пример аналогичен второму примеру за исключением того, что в переменные *bnd1*, *bnd2* и *bnd3* вызывающей команду программы записываются четверть нового дохода, фамилия и номер отдела соответственно.

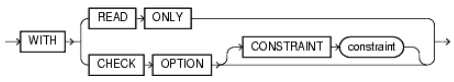


Команда *DELETE* предназначена для удаления строк таблиц и представлений.

table_expression_clause определяет объект БД (обычно, таблицу), откуда удаляются записи. Вместо прямого указания на объект БД можно использовать подзапрос к нему. Это обеспечивает дополнительную функциональность.

Команда DELETE (продолжение)

with_clause::=



where_clause::=

returning_clause::=

with_clause используется в командах *DELETE* с подзапросом в качестве *table_expression_clause*, указывается в подзапросе и позволяет ограничить результат выполнения команды одним из следующих способов:

- *WITH READ ONLY* указывает, что таблица или представление не будут изменяться;
- *WITH CHECK OPTION* указывает, чтобы Oracle запрещал всякие изменения таблицы или представления, которые приведут к появлению строк, не удовлетворяющих подзапросу.

WITH CHECK OPTION CONSTRAINT позволяет задать имя «*WITH CHECK OPTION*»-ограничения (в противном случае ему будет присвоено системное имя).

where_clause определяет условие фильтрации строк изменяемого объекта БД (таблицы или представления). Удалению будут подвергнуты только строки, для которых это условие даст значение «*истина*». Если условие отсутствует, удалятся все строки объекта.

returning_clause позволяет возвратить в вызвавшую команду программу значения выражений, включающих значения столбцов удаляемых строк. Список этих выражений указывается после слова *RETURNING*, а следом после слова *INTO* задаются имена переменных, объявленных в программе. Количество и типы выражений и переменных должны соответствовать друг другу.

SQL. Примеры запросов. DELETE

```
DELETE FROM temp_assign;
DELETE FROM emp
    WHERE JOB = 'SALESMAN'    AND COMM < 100;
DELETE FROM (select * from emp)
    WHERE JOB = 'SALESMAN'    AND COMM < 100;
DELETE FROM emp
    WHERE ename = 'JONES' RETURNING sal INTO :bnd1;
```

Примеры, демонстрирующие использование команды *DELETE*, незамысловаты.

Первый запрос полностью очищает таблицу *temp_assign*. Второй удаляет строки служащих с должностью продавец (*SALESMAN*), имеющих комиссионные меньше 100.

Второй запрос делает абсолютно то же самое, но вместо таблицы использует подзапрос.

Третий запрос удаляет строку о служащем по фамилии *JONES* и записывает в переменную *bnd1* его доход.

Вопросы и задания к пункту 4.2.4

1. Перечислите классы спецификационных языков реляционной модели данных.
2. Дайте определения основных и дополнительных операций реляционной алгебры Кодда. Поясните на примерах их работу.
3. Что представляет собой запрос в реляционном исчислении с переменными-кортежами?
4. Какие атомарные выражения используются в реляционном исчислении с переменными-кортежами?
5. По каким правилам строятся формулы в реляционном исчислении с переменными-кортежами?
6. Что представляет собой запрос в реляционном исчислении с переменными на доменах?
7. Какие атомарные выражения используются в реляционном исчислении с переменными на доменах?
8. По каким правилам строятся формулы в реляционном исчислении с переменными на доменах?
9. Назовите особенности языка QBE.
10. Охарактеризуйте три типа синтаксических конструкций языка SQL, начинающихся с ключевого слова SELECT. Для чего они предназначены? В чем особенности каждой конструкции?
11. Какова грамматика и семантика табличного выражения команды SELECT?