

Технологии баз данных

7. Манипулирование данными в реляционной модели.
Реляционная алгебра
8. Манипулирование данными в реляционной модели.
Реляционное исчисление
9. SQL

Технологии баз данных

Тема 7. Манипулирование данными в
реляционной модели. Реляционная
алгебра

Манипулирование данными в реляционной модели

Для манипулирования данными в реляционной модели используются два формальных аппарата:

- **реляционная алгебра**, основанная на теории множеств;
- **реляционное исчисление**, базирующееся на исчислении предикатов первого порядка.

*Операции, реализуемые с помощью указанных аппаратов, обладают важным свойством: **они замкнуты на множестве отношений.***

Манипулирование данными в реляционной модели

Конкретный язык манипулирования реляционными БД называется реляционно полным, если любой запрос, выражаемый с помощью одного выражения реляционной алгебры или одной формулы реляционного исчисления, может быть выражен с помощью одного оператора этого языка.

Заметим, что крайне редко алгебра или исчисление принимаются в качестве полной основы какого-либо языка БД. Обычно (как, например, в случае языка SQL) язык основывается на некоторой смеси алгебраических и логических конструкций.

Реляционная алгебра

Операции реляционной алгебры определены на множестве отношений и являются замкнутыми относительно этого множества (образуют алгебру).

*В состав теоретико-множественных операций входят **традиционные операции над множествами**:*

- объединение;*
- пересечение;*
- разность;*
- декартово произведение.*

***Специальные реляционные операции** включают:*

- выборку;*
- проекцию;*
- естественное соединение;*
- деление (частное).*

*Операции **объединения, пересечения и разности** применяются к отношениям совместимым по типу, или другими словами к **отношениям с эквивалентными схемами**.*

Реляционная алгебра

Объединением двух совместимых по типу отношений R и S ($R \cup S$) называется отношение с тем же заголовком, как в отношениях R и S , и с телом, состоящим из множества кортежей t , принадлежащих R или S или обоим отношениям.

Отношение R

Отношение S

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>AGE</i>
1809	Иванов	Москва	45
1996	Петров	Нижний Новгород	39
1777	Сидоров	Рязань	21

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>AGE</i>
1809	Иванов	Москва	45
1896	Галкин	Иваново	40

$R \cup S$

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>AGE</i>
1809	Иванов	Москва	45
1996	Петров	Нижний Новгород	39
1777	Сидоров	Рязань	21
1896	Галкин	Иваново	40

Реляционная алгебра

Пересечением двух совместимых по типу отношений R и S ($R \cap S$) называется отношение с тем же заголовком, как в отношениях R и S , и с телом, состоящим из множества кортежей t , принадлежащих одновременно обоим отношениям R и S .

Отношение R

Отношение S

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>AGE</i>
1809	Иванов	Москва	45
1996	Петров	Нижний Новгород	39
1777	Сидоров	Рязань	21

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>AGE</i>
1809	Иванов	Москва	45
1896	Галкин	Иваново	40

$R \cap S$

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>AGE</i>
1809	Иванов	Москва	45

Реляционная алгебра

Разностью двух совместимых по типу отношений R и S ($R - S$) называется отношение с тем же заголовком, как в отношениях R и S , и с телом, состоящим из множества кортежей t , принадлежащих отношению R и не принадлежащих отношению S .

Отношение R

Отношение S

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>AGE</i>
1809	Иванов	Москва	45
1996	Петров	Нижний Новгород	39
1777	Сидоров	Рязань	21

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>AGE</i>
1809	Иванов	Москва	45
1896	Галкин	Иваново	40

$R - S$

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>AGE</i>
1996	Петров	Нижний Новгород	39
1777	Сидоров	Рязань	21

Реляционная алгебра

R1(ФИО, Паспорт, Школа)

R2(ФИО, Паспорт, Школа)

R3(ФИО, Паспорт, Школа)

1. Список абитуриентов, которые поступали два раза, но так и не поступили в вуз.
2. Список абитуриентов, которые поступили в вуз с первого раза.
3. Список абитуриентов, которые поступили в вуз со второго раза.
4. Список абитуриентов, которые поступали в вуз один раз и не

Реляционная алгебра

Декартово произведение двух отношений R и S ($R \times S$), определяется как отношение с заголовком, представляющим собой сцепление двух заголовков исходных отношений R и S , и телом, состоящим из множества кортежей t , таких, что первым является любой кортеж отношения R , а вторым – любой кортеж, принадлежащий отношению S .

Отношение R

S1
S2
S3

Отношение S

P1
P2
P3
P4

$R \times S$

S1	P1
S1	P2
S1	P3
S1	P4

S2	P1
S2	P2
S2	P3
S2	P4

S3	P1
S3	P2
S3	P3
S3	P4

Реляционная алгебра

Выборка — это сокращенное название θ - выборки, где θ означает любой скалярный оператор сравнения ($\neq, \leq, \geq, =$).

$$\sigma_{X\theta Y}(R)$$

θ - выборкой, из отношения R по атрибутам X и Y называется отношение, имеющее тот же заголовок, что и отношение R , и тело, содержащее множество кортежей t отношения R , для которых проверка условия $X \theta Y$ дает значение истина. Атрибуты X и Y должны быть определены на одном и том же домене, а оператор должен иметь смысл для этого домена.

Отношение R

ID_NUM	NAME	CITY	AGE
1809	Иванов	Москва	45
1996	Петров	Нижний Новгород	39
1777	Сидоров	Рязань	21
1896	Галкин	Москва	30

$$\sigma_{CITY='Москва'}(R)$$

ID_NUM	NAME	CITY	AGE
1809	Иванов	Москва	45
1896	Галкин	Москва	30

Реляционная алгебра

Проекцией отношения R по атрибутам X, Y, ..., Z ($P_{[X, Y, \dots, Z]}(R)$), где каждый из атрибутов принадлежит отношению R, называется отношение с заголовком {X, Y, ..., Z} и с телом, содержащим множество всех кортежей вида $\langle X:x, Y:y, \dots, Z:z \rangle$ таких, что в отношении R имеется кортеж, атрибут X которого имеет значение x, атрибут Y имеет значение y, ..., атрибут Z имеет значение z.

Отношение R

ID_NUM	NAME	CITY	AGE
1809	Иванов	Москва	45
1996	Петров	Нижний Новгород	39
1777	Сидоров	Рязань	21
1896	Галкин	Москва	30

$P_{[NAME, CITY]}(R)$

<i>NAME</i>	<i>CITY</i>
Иванов	Москва
Петров	Нижний Новгород
Сидоров	Рязань
Галкин	Москва

$P_{[CITY]}(R)$

<i>CITY</i>
Москва
Нижний Новгород
Рязань

Реляционная алгебра

Соединение отношений — создает новое отношение, каждый кортеж которого является результатом сцепления кортежей операндов (исходных отношений).

Соединение имеет две разновидности: ***естественное соединение и соединение по условию (θ -соединение)***.

Пусть $X=\{X_1, X_2, \dots, X_m\}$, $Y=\{Y_1, Y_2, \dots, Y_n\}$, $Z=\{Z_1, Z_2, \dots, Z_k\}$.

Естественным соединением отношений $R(X,Y)$ и $S(Y,Z)$ называется отношение с заголовком $\{X, Y, Z\}$ и с телом, содержащим множество всех кортежей вида $\langle X:x, Y:y, Z:z \rangle$ таких, для которых в отношении R значение атрибута X равно x , а значение атрибута Y равно y , и в отношении S значение атрибута Y равно y , а атрибута Z равно z .

При естественном соединении производится сцепление строк операндов соединения по общим атрибутам при условии равенства общих атрибутов.

Замечание 1. Соединения не всегда выполняются по внешнему ключу и соответствующему первичному ключу, хотя такие соединения очень распространены и являются важным частным случаем.

Замечание 2. Если отношения R и S не имеют общих атрибутов, то выражение $R \bowtie S$ эквивалентно $R \times S$.

Реляционная алгебра

Соединение отношений

Отношение *R* (поставщики)

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>STATUS</i>
1809	Иванов	Москва	20
1996	Петров	Нижний Новгород	15
1777	Сидоров	Рязань	10

Отношение *S* (детали)

<i>IP_NUM</i>	<i>NAMEN</i>	<i>CITY</i>	<i>WEIGHT</i>
P123	Болт	Москва	12
P896	Гайка	Нижний Новгород	14
P432	Шарнир	Москва	15

$R \bowtie S$

<i>ID_NUM</i>	<i>NAME</i>	<i>STATUS</i>	<i>CITY</i>	<i>IP_NUM</i>	<i>NAMEN</i>	<i>CITY</i>	<i>WEIGHT</i>
1809	Иванов	20	Москва	P123	Болт	Москва	12
1809	Иванов	20	Москва	P432	Шарнир	Москва	15
1996	Петров	15	Нижний Новгород	P896	Гайка	Нижний Новгород	14

Реляционная алгебра

θ -соединение

Пусть отношения R и S не имеют общих имен атрибутов, и θ определяется так же, как в операции выборки.

θ - соединением отношения R по атрибуту X с отношением S по атрибуту Y называется результат вычисления выражения

$$\sigma_{X\theta Y}(R \times S)$$

θ -соединение — это отношение с тем же заголовком, что и при декартовом произведении отношений R и S , и с телом, содержащим множество кортежей $t \in R \times S$, таких, что вычисление условия $X \theta Y$ дает значение истина для данного кортежа.

Атрибуты X и Y должны быть определены на одном и том же домене, а оператор должен иметь смысл для этого домена.

Реляционная алгебра

θ -соединение

Отношение R (поставщики)

<i>ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>STATUS</i>
1809	Иванов	Москва	20
1996	Петров	Нижний Новгород	15
1777	Сидоров	Рязань	10

Отношение S (поставки)

<i>ID_NUM</i>	<i>IP_NUM</i>	<i>QTY</i>
1809	P123	100
1809	P896	200
1777	P432	150
1996	P432	150
1996	P123	250

$\sigma_{QTY < 200 \text{ and } R.ID_NUM = S.ID_NUM} (R \times S)$

<i>R.ID_NUM</i>	<i>NAME</i>	<i>CITY</i>	<i>STATUS</i>	<i>S.ID_NUM</i>	<i>IP_NUM</i>	<i>QTY</i>
1809	Иванов	Москва	20	1809	P123	100
1996	Петров	Нижний Новгород	15	1996	P432	150
1777	Сидоров	Рязань	10	1777	P432	150

Реляционная алгебра

Операция деления

У операции реляционного деления два операнда - бинарное и унарное отношения. Пусть $X=\{X_1, X_2, \dots, X_m\}$, $Y=\{Y_1, Y_2, \dots, Y_n\}$.

Делением отношений $R(X,Y)$ на $S(Y)$ (**R/S**) называется отношение с заголовком $\{X\}$ и телом, содержащим множество всех кортежей $\{X:x\}$, таких, что существует кортеж $\{X:x, Y:y\}$, который принадлежит отношению **R для всех кортежей $\{Y:y\}$, принадлежащих отношению S .**

Деление отношений — создает новое отношение, содержащее атрибуты первого отношения, отсутствующие во втором отношении и кортежи, которые при сцеплении с кортежами второго отношения, будут принадлежать первому отношению. Для выполнения этой операции второе отношения должно содержать лишь атрибуты, совпадающие с атрибутами первого.

Реляционная алгебра

Операция деления

Отношение A
Отношение B2

<i>S#</i>	<i>P#</i>
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P3
S3	P2
S3	P3

Отношение B

<i>P#</i>
P1

<i>A/B</i>
S1
S2

Отношение B1

<i>P#</i>
P2
P3

<i>A/B1</i>
S1
S3

<i>P#</i>
P1
P2
P3

<i>A/B2</i>
S1

Реляционная алгебра

R1(ФИО, Дисциплина, Оценка)

R2(ФИО, Группа)

R3(Группа, Дисциплина)

1. Список студентов сдавших БД на отлично.
2. Список студентов, кто должен был сдавать экзамен по БД, но ещё не сдавал.
3. Список студентов, которые имеют несколько двоек.
4. Список круглых отличников.

Технологии баз данных

Тема 8. Манипулирование данными в
реляционной модели.
Реляционное исчисление

Реляционное исчисление

Базисными понятиями исчисления являются понятие переменной с определенной для нее областью допустимых значений и понятие правильно построенной формулы, опирающейся на переменные, предикаты и кванторы.

В логике первого порядка (или теории исчисления предикатов) под **предикатом** подразумевается **истинностная функция с параметрами**. После подстановки значений вместо параметров функция становится выражением, называемым **суждением**, которое может быть истинным или ложным.

Пример предиката:

X является сотрудником

организации

X имеет более высокую зарплату,

чем Y 1. Реляционное исчисления кортежей (Кодд)

2. Реляционное исчисления доменов (Лякруа и Пиротт).

$$\{x \mid P(x)\}$$

Реляционное исчисление с переменными кортежами

Областями определения переменных являются отношения базы данных, т.е. допустимым значением каждой переменной является кортеж некоторого отношения.

Формулы в реляционном исчислении кортежей имеют вид $\{t \mid \psi(t)\}$

где t — переменная - кортеж, т.е. переменная, обозначающая кортеж некоторой фиксированной длины

ψ — формула, построенная из атомов и совокупности операторов

Атомы формул ψ могут быть трех типов:

$$R(t) \quad t[i] \Theta s[j] \quad t[i] \Theta const$$

При определении операций реляционного исчисления введем понятия «**свободных**» и «**связанных**» переменных - кортежей.

Неформально вхождение переменной в формулу является «**связанным**», если этой переменной предшествует квантор «для всех» - всеобщности или

«существует» -

существования.

В противном случае переменная называется «**свободной**».

Реляционное исчисление с переменными кортежами

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 $\neg \psi_1$ также 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 являются свободными или связными в $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$ и $\neg \psi_1$ точно так же, как они являются свободными или связными в ψ_1 и ψ_2 , в зависимости, где они появляются.
3. Если ψ - формула, то $(\exists t)(\psi)$ - также формула. Вхождения переменной t , свободные в формуле ψ , являются связанными квантором $(\exists t)$ в формуле $(\exists t)(\psi)$. Формула $(\exists t)(\psi)$ утверждает, что существует значение t , при подстановке которого вместо всех свободных вхождений t в формулу ψ эта формула становится истинной. Например, $(\exists t)(R(t))$ означает, что отношение R не пусто, т.е. существует некоторый кортеж t , принадлежащий R .
4. Если ψ — формула, то и $(\forall t)(\psi)$ — тоже формула. Свободные вхождения t в ψ становятся связанными квантором $(\forall t)$ в $(\forall t)(\psi)$. Другие вхождения переменных в ψ интерпретируются так же, как в п.3. Формула $(\forall t)(\psi)$ утверждает, что, какое бы значение подходящей арности не подставлялось вместо свободных вхождений t в ψ , формула ψ станет истинной.
5. Формулы могут при необходимости заключаться в скобки. Предполагается следующий порядок старшинства: арифметические операции сравнения, кванторы \exists и \forall и, наконец, \neg , \wedge , \vee в перечисленном порядке.
6. Ничто другое не является формулой.

Реляционное исчисление с переменными кортежами

Квантор существования используется для указания количества экземпляров, к которым должен быть применен предикат, и используется в формуле, которая должна быть истинной хотя бы для одного экземпляра, например:

$$\{t \mid R(t) \wedge (\exists s) (S(s) \wedge (s[ID] = t[ID]) \wedge s[FIO] = \text{'Иванов'})\}$$

Это выражение означает, что в отношении S существует кортеж, который имеет такое же значение атрибута ID , что и значение атрибута ID в текущем кортеже t из отношения R , а атрибут FIO из кортежа s имеет значение 'Иванов'.

Квантор всеобщности используется в выражениях, которые относятся ко всем экземплярам, например:

$$\{t \mid (\forall t) (t[CITY] \neq \text{'Харьков'})\}$$

Это выражение означает, что ни в одном кортеже отношения R значение атрибута $CITY$ не равно 'Харьков'.

В отношении логических операций могут применяться следующие правила эквивалентности:

1. $(\exists t) (\psi(t)) \equiv \neg(\forall t) (\neg(\psi(t)))$
2. $(\forall t) (\psi(t)) \equiv \neg(\exists t) (\neg(\psi(t)))$
3. $(\exists t) (\psi_1(t) \wedge (\psi_2(t))) \equiv \neg(\forall t) (\neg(\psi_1(t)) \vee \neg(\psi_2(t)))$
4. $(\forall t) (\psi_1(t) \wedge (\psi_2(t))) \equiv \neg(\exists t) (\neg(\psi_1(t)) \vee \neg(\psi_2(t)))$

Исходя из рассмотренных правил, последнюю формулу можно записать следующим образом:

$$\{t \mid \neg(\exists t) (t[CITY] = \text{'Харьков'})\}$$

Реляционное исчисление с переменными кортежами

$$\{t \mid R(t) \wedge \neg S(t)\}$$

$$\{t^{(r+p)} \mid (\exists u^{(r)}) (\exists v^{(p)}) (R(u) \wedge S(v) \wedge t[1]=u[1] \wedge t[2]=u[2] \wedge \dots \wedge t[r]=u[r] \wedge t[r+1]=v[1] \wedge t[r+2]=v[2] \wedge \dots \wedge t[r+p]=v[p])\}$$

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

$$\{t \mid (\exists u) (R(u) \wedge F')\}$$

R1(ФИО, Дисциплина, Оценка)

R2(ФИО, Группа)

R3(Группа, Дисциплина)

Реляционное исчисление с переменными доменами

Областями определения переменных являются домены на которых определены атрибуты отношений БД. *Допустимым значением* каждой переменной является *значение некоторого домена*.

Формула имеет вид $\{ t_1, t_2, \boxtimes, t_k \mid \psi(t_1, t_2, \boxtimes, t_k) \}$

Где t_1, t_2, \boxtimes, t_k переменные на доменах

$\psi(t_1, t_2, \boxtimes, t_k)$ формула, построенная из атомов

Атомы формулы ψ могут быть трех

типов:

$$R(t_1, t_2, \boxtimes, t_k) \quad t_i \ominus t_j \quad t_i \ominus const$$

Формула строится по тем же правилам с использованием логических операций и кванторов всеобщности и существования.

Реляционное исчисление с переменными доменами

$$\{ t_1, t_2 \mid R_1(t_1, t_2) \wedge (\forall y)(\neg R_2(t_1, y) \wedge \neg R_2(t_2, y)) \}$$

Реляционное исчисление с переменными доменами

Правила перехода от переменных кортежей к переменным доменам.

1. Если \mathbf{t} кортеж арности k , то вводятся t_1, t_2, \dots, t_k переменных на доменах
2. Атом $R(\mathbf{t})$ заменяется атомом $R(t_1, t_2, \dots, t_k)$
3. Свободное вхождение $\mathbf{t}[i]$ заменяется на t_i
4. Для каждого квантора Q , в области действия, выполняется замена

$$(\forall \mathbf{t}) \quad (\forall t_1)(\forall t_2) \dots (\forall t_k)$$

$$(\exists \mathbf{t}) \quad (\exists t_1)(\exists t_2) \dots (\exists t_k)$$

Реляционное исчисление с переменными доменами

$R \times S$

$$\{t_1, t_2, \dots, t_{r+s} \mid (\exists u_1) \dots (\exists u_r)(\exists v_1) \dots (\exists v_s) \\ (R(u_1, \dots, u_r) \wedge S(v_1, \dots, v_s) \wedge \\ t_1 = u_1 \wedge \dots \wedge t_r = u_r \\ \wedge t_{r+1} = v_1 \wedge \dots \wedge t_{r+s} = v_s)\}$$

Языки манипулирования данными

Реляционная алгебра

ISBL (Information Systems Base Language)

IBM (Питерли, Англия)

экспериментальная система PRTV (Peterlee Relational Test Vehicle)

Comparison: Relational Algebra vs ISBL

$R \cup S$	$R+S$
$R - S$	$R-S$ subsumes
$R \times S$	no direct counterpart
$\prod_{i(1), i(2), \dots, i(t)} (R)$	$R \% A, B, \dots, Z$
$\sigma_F(R)$	$R : F$
contrived derived op	$R * S$

To prove completeness of ISBL, enough to show that can express Cartesian product using the ISBL operators - return to this issue later

Example ISBL query to specify the composition of two binary relations $R(A,B)$ and $S(C,D)$ where A,B,C,D are attributes defined over the same domain X (as when defining composition of functions $X \square X$):

Specify composition of R and S as RCS , where

$$RCS = (R * S) : B=C \% A, D$$

In this case: $R * S = R \times S$ because attribute names $(A, B), (C, D)$ are disjoint [cf. completeness of ISBL]

Illustrates archetypal form of query definition:

projection of selection of join

$(R \% A, B \rightarrow D) * S$ прямое декартово произведение ($B \rightarrow D$ переименование)

$$RCS = N!R * N!S : B = C \% A, D$$

Языки манипулирования данными

Реляционное исчисление с кортежами

QUEL University of California, Berkeley СУБД Ingres используется с конца 70-х годов

Основной набор операторов манипулирования данными включает операторы **RETRIVE (выбрать)**, **APPEND (добавить)**, **REPLACE (заменить)** и **DELETE (удалить)**. Перед выполнением любого из этих операторов необходимо определить используемые в них переменные кортежей, связав их с соответствующими отношениями путем выполнения оператора **RANGE**.

RANGE OF variable-list IS relation-name

RANGE OF S IS СТУДЕНТЫ

RANGE OF G IS ГРУППЫ

RETRIEVE (S.СТУД_ИМЯ)

WHERE (S.ГРУП_НОМЕР = G.ГРУП_НОМЕР AND G.КУРАТ_ИМЯ = "ИВАНОВ")

REPLACE S(СТУД_СТИП BY СТУД_СТИП * 1,5) WHERE (S.СТУД_УСП = "YES")

RETRIEVE (S.СТУД_ИМЯ) WHERE (S.СТУД_СТИП < AVG (S.СТУД_СТИП))

Выбрать имена студентов, куратором которых является Иванов.

Языки манипулирования данными

Реляционное исчисление с кортежами

POSTQUEL (англ. Postgres Query Language) – первичный язык запросов для СУБД Postgres, в настоящее время PostgreSQL.

Этот язык был разработан в 1985 году в Калифорнийском университете Беркли командой разработчиков, работающих под руководством профессора Майкла Стоунбрейкера. POSTQUEL основывается на языке запросов QUEL.

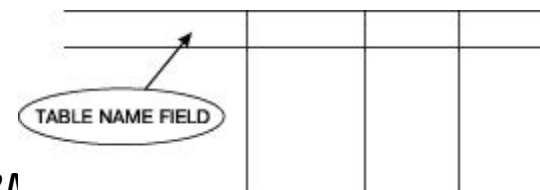
В 1995 г. Эндрю Ю (Andrew Yu) и Джолли Чен (Jolly Chen) заменили в базе Postgres POSTQUEL язык запросов на SQL.

Языки манипулирования данными

Реляционное исчисление на домене

Query by Example (QBE) "Запрос по образцу"

Разработан Мойше Злуфом в 1974-1975 гг. в фирме IBM.



- I. (insert) — включить;
- D. (delete) — удалить;
- U. (update) — обновить;
- P. (print) — печатать.

запрос QBE

dept	deptno	dname	loc
	P.	SALES	

```
SELECT deptno FROM dept WHERE  
dname='SALES'
```

Таблица 9.12. Вставка строки

emp	ename	mgr	sal	deptno
I.	JONES	7638	43550	40

Таблица 9.14. Удаление

сотрудников отдела продаж

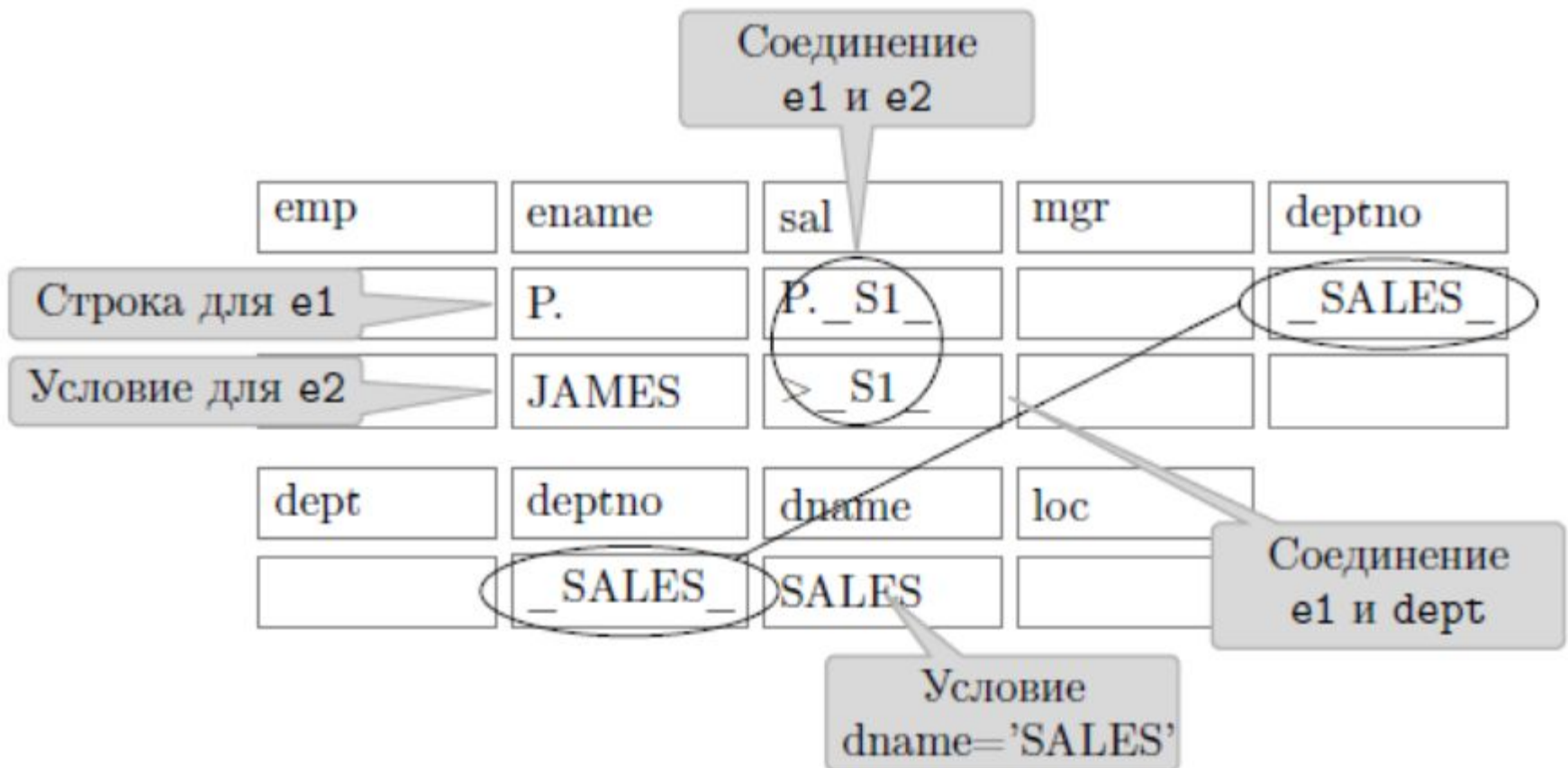
emp	ename	mgr	sal	deptno
D.				_D_
dept	deptno	dname	loc	
	D	SALES		

Основное назначение переменных
— создание соединений таблиц.

Языки манипулирования данными

Реляционное исчисление на домене Query by Example (QBE)

Соединение таблицы emp с собой и с таблицей dept в запросе: "Найти имена и зарплаты служащих, получающих больше, чем JAMES, и работающих в отделе продаж (SALES)"



Языки манипулирования данными

Данный способ создания запросов позволяет получить высокую наглядность и не требует указывать алгоритм выполнения операции. Многие современные реляционные СУБД содержат свой вариант **QBE**.

Запрос2 : запрос на выборку

Группы: КодГруппы, Название, Курс, Семестр

Студенты: КодСтудент, КодГруппы, Фамилия, Имя

Успеваемость: КодОценки, КодДисциплины, КодСтудент, Оценка

Дисциплины: КодДисциплины, Название, Кол часов

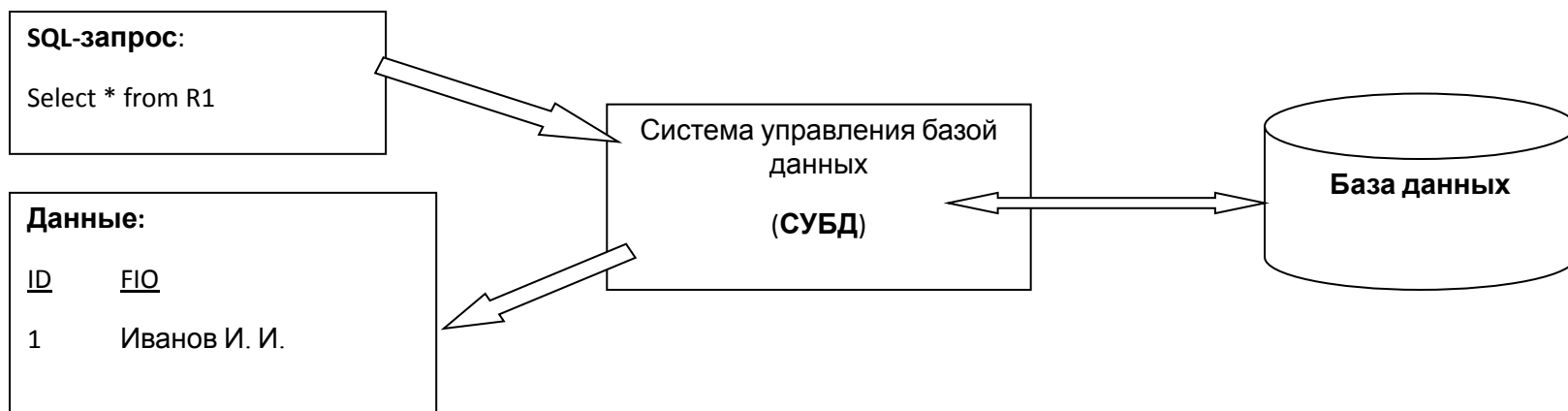
Поле:	Название	Фамилия	Оценка	Название
Имя таблицы:	Группы	Студенты	Успеваемость	Дисциплины
Сортировка:				
Вывод на экран:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Условие отбора:			"отл/А"	
или:				

Технологии баз данных

Тема 9. SQL

SQL — Structured Query Language

Разработан в **1974** году фирмой **IBM** для экспериментальной реляционной СУБД **System R**.



SQL является достаточно мощным языком, обеспечивающим эффективное взаимодействие с СУБД.

SQL на сегодняшний день является единственным стандартным языком для работы с реляционными БД.

SQL – это не полноценный компьютерный язык типа C++ или Java.

SQL – это слабоструктурированный язык.

SQL — Structured Query Language

Используется для:

□ Организация данных.

SQL дает пользователю возможность изменять структуру представления данных, а также устанавливать отношения между элементами БД.

□ Выборка данных.

SQL дает пользователю возможность изменять БД, т.е. добавлять в неё новые данные, а также удалять или обновлять уже имеющиеся в ней данные.

□ Управление доступом.

С помощью SQL можно ограничить возможности пользователя по выборке и изменению данных и защитить их от несанкционированного доступа.

□ Совместное использование данных.

SQL координирует совместное использование данных пользователями, работающими параллельно, чтобы они не мешали друг другу.

□ Целостность данных.

SQL позволяет обеспечить целостность БД, защищая её от разрушения из-за несогласованных изменений или отказа системы.

SQL — Structured Query Language

Достоинства SQL.

SQL – это легкий для понимания язык и в тоже время универсальное программное средство управления данными.

Успех языку SQL принесли следующие его особенности:

- ✓ независимость от конкретной СУБД;
- ✓ межплатформенная переносимость;
- ✓ наличие стандартов;
- ✓ реляционная основа;
- ✓ поддержка со стороны компаний IBM(СУБД DB2) и Microsoft (протокол ODBC и технология ADO);
- ✓ возможность выполнения специальных интерактивных запросов;
- ✓ поддержка архитектуры клиент/сервер;
- ✓ возможность различного представления данных;
- ✓ интеграция с языками высокого уровня;
- ✓ расширяемость и поддержка объектно-ориентированных технологий.

SQL — Structured Query Language

Недостатки SQL

✓ *Несоответствие реляционной модели данных*

Создатели реляционной модели данных Эдгар Кодд, Кристофер Дейт и их сторонники указывают на то, что SQL не является истинно реляционным языком. В частности, они указывают на следующие проблемы SQL:

1. повторяющиеся строки;
2. неопределённые значения (NULL);
3. явное указание порядка колонок слева направо;
4. колонки без имени и дублирующиеся имена колонок;
5. отсутствие поддержки свойства «=»;
6. использование указателей;
7. высокая избыточность.

▣ *Сложность*

Хотя SQL и задумывался как средство работы конечного пользователя, в конце концов, он стал настолько сложным, что превратился в инструмент программиста.

Недостатки SQL

□ Отступления от стандартов

Несмотря на наличие международного стандарта SQL, многие компании, занимающиеся разработкой СУБД (например, Oracle, Sybase, Microsoft, MySQL AB), вносят изменения в язык SQL, применяемый в разрабатываемой СУБД, тем самым отступая от стандарта. Таким образом, появляются специфичные для каждой конкретной СУБД диалекты языка SQL.

□ Сложность работы с иерархическими структурами

Ранее диалекты SQL большинства СУБД не предлагали способа манипуляции древовидными структурами. Некоторые поставщики СУБД предлагали свои решения (например, Oracle использует выражение CONNECT BY). В настоящее время в ANSI стандартизована рекурсивная конструкция WITH из диалекта SQL DB2. В MS SQL Server рекурсивные запросы появились лишь в версии MS SQL Server 2005. В версии MS SQL Server 2008 появился новый тип данных — hierarchyid, упрощающий манипуляцию древовидными структурами.

SQL — Structured Query Language

Data Definition Language (DDL).

Оператор	Описание
<i>CREATE</i>	Применяется для добавления нового объекта к базе данных
<i>DROP</i>	Применяется для удаления объекта из базы данных
<i>ALTER</i>	Применяется для изменения структуры имеющегося объекта

Data Manipulation Language (DML).

Оператор	Описание
<i>SELECT</i>	Применяется для выбора данных
<i>INSERT</i>	Применяется для добавления строк к таблице
<i>DELETE</i>	Применяется для удаления строк из таблицы
<i>UPDATE</i>	Применяется для изменения данных

Transaction Control Language (TCL).

Оператор	Описание
<i>COMMIT</i>	Применяется для завершения транзакции и сохранения изменений в БД
<i>ROLLBACK</i>	Применяется для отката транзакции и отмены изменений в БД
<i>SET TRANSACTION</i>	Применяется для установки параметров доступа к данным в текущей транзакции

Data Control Language (DCL).

Оператор	Описание
<i>GRANT</i>	Применяется для присвоения привилегии
<i>DENY</i>	Запрещает некоторую привилегию
<i>REVOKE</i>	Применяется для отмены привилегии

SQL — Structured Query Language

```
SELECT [[ALL] | DISTINCT] [TOP n [PERCENT]] [WITH TIES]
{* | элемент_SELECT [, элемент_SELECT] ...}
FROM   таблица [псевдоним] [, таблица [псевдоним]] ...
WHERE  условие_отбора_строк ]
GROUP BY [таблица.]столбец [, [таблица.]столбец] ...
HAVING  условие_отбора_групп]]
ORDER BY{[таблица.]столбец | номер_элемента_SELECT}
[[ASC] | DESC]
[, {[таблица.]столбец | номер_элемента_SELECT } [ [ASC] |
DESC] ] ...] );
```

SQL — Structured Query Language

Этот оператор можно прочитать следующим образом:

SELECT (выбрать) — данные из указанных столбцов и (если необходимо) выполнить перед выводом их преобразование в соответствии с указанными выражениями и (или) функциями

FROM (из) — перечисленных таблиц, в которых расположены эти столбцы

WHERE (где) — строки из указанных таблиц должны удовлетворять указанному перечню условий отбора строк

SQL — Structured Query Language

GROUP BY (группируя по) — указанному перечню столбцов с тем, чтобы получить для каждой группы единственное агрегированное значение, используя во фразе SELECT SQL-функции SUM (сумма), COUNT (количество), MIN (минимальное значение), MAX (максимальное значение) или AVG (среднее значение)

HAVING (имея) — в результате лишь те группы, которые удовлетворяют указанному перечню условий отбора групп

ORDER BY (упорядочить) — результаты выбора данных по указанному перечню столбцов. При этом упорядочение можно производить в порядке возрастания - ASC (ASCending)(по умолчанию) или убывания DESC (DESCending).

SQL — Structured Query Language

Параметры раздела обозначают следующее:

ALL – указывает, что в результат выборки должны быть включены все строки возвращаемые запросом, т.е. выборка может содержать повторяющиеся строки (используется по умолчанию).

DISTINCT – позволяет исключить из выборки повторяющиеся строки.

TOP n [PERCENT] [WITH TIES] – ограничивает количество строк в выборке. Параметр n задает максимальное количество строк, при указании параметра PERCENT количество строк задается в процентах от общего числа строк, возвращаемых запросом.

*- означает включение в результат выборки всех столбцов всех таблиц, участвующих в запросе и указанных в разделе FROM. При этом порядок вывода полей соответствует порядку, в котором эти поля определялись при создании

SQL — Structured Query Language

Параметры раздела обозначают следующее:

элемент_SELECT – список столбцов, которые включены в результат выборки.

Структура этой конструкции следующая:

элемент_SELECT = {[таблица.]* | [таблица.]столбец [AS псевдоним] | (выражение) [AS псевдоним] | константа [AS псевдоним] | переменная [AS псевдоним] | SQL_функция [AS псевдоним]}

термин **таблица** – используется для обобщения понятий: базовая таблица, представление.

выражение – подразумевает выражение, на основе которого будет формироваться содержимое столбца.

AS псевдоним – определение псевдонима для столбца.

Выборка с использованием фразы WHERE

Раздел **WHERE** предназначен для ограничения количества строк, включаемых в результат выборки. Будут включены только те строки, которые удовлетворяют условию отбора строк.

WHERE условие_отбора_строк

где условие_отбора_строк – выражение логического типа (TRUE, FALSE).

В условии можно использовать операторы сравнения = (равно), <> (не равно), < (меньше), <= (меньше или равно), > (больше), >= (больше или равно), которые могут предваряться оператором NOT, создавая, например, отношения "не меньше" и "не больше".

условие_отбора_строк - предназначено для объединения множества логических условий, каждое из которых возвращает выражение логического типа. Объединение⁵¹

Выборка с использованием фразы WHERE

R1(ФИО, Дисциплина, Оценка)

R2(ФИО, Группа)

R3(Группа, Дисциплина)

SELECT ФИО

FROM R2

WHERE группа LIKE 'ПМИ-3[12]БО'

ГРУППА IN ('ПМИ-31БО','ПМИ-32БО')

Объединение

<запрос 1>

UNION [ALL]

<запрос 2>

Предложение **UNION** приводит к появлению в результирующем наборе всех строк каждого из запросов. При этом, если определен параметр **ALL**, то сохраняются все дубликаты выходных строк, в противном случае в результирующем наборе присутствуют только уникальные строки. Заметим, что можно связывать вместе любое число запросов. Кроме того, с помощью скобок можно задавать порядок объединения.

Операция объединения может быть выполнена только при выполнении следующих условий:

- количество выходных столбцов каждого из запросов должно быть одинаковым;
- выходные столбцы каждого из запросов должны быть совместимы между собой (в порядке их следования) по типам данных;
- в результирующем наборе используются имена столбцов, заданные в первом запросе;
- предложение **ORDER BY** применяется к результату объединения, поэтому оно может быть указано только в конце всего составного запроса.

Пересечение и разность

INTERSECT [ALL] (пересечение)

EXCEPT [ALL] (разность)

В результирующий набор попадают только те строки, которые присутствуют в обоих запросах (**INTERSECT**) или только те строки первого запроса, которые отсутствуют во втором (**EXCEPT**). При этом оба запроса, участвующих в операции, должны иметь одинаковое число столбцов, и соответствующие столбцы должны иметь одинаковые (или неявно приводимые) типы данных. Имена столбцов результирующего набора формируются из заголовков первого запроса.

Если не используется ключевое слово **ALL** (по умолчанию подразумевается **DISTINCT**), то при выполнении операции автоматически устраняются дубликаты строк.

Если указано **ALL**, то количество дублированных строк подчиняется следующим правилам (n_1 - число дубликатов строк первого запроса, n_2 - число дубликатов строк второго запроса):

INTERSECT ALL: $\min(n_1, n_2)$

EXCEPT ALL: $n_1 - n_2$, если $n_1 > n_2$.

NULL-значения в выражениях.

Как правило, применение NULL-значения в выражении приводит к результату, равному NULL.

Например, `SELECT (5+NULL)` вернет NULL, а не 5. Как и в случае простых выражений, при передаче большинству функций NULL-значений результатом будет NULL.

Возможность неопределенных значений в реляционных базах данных означает, что для любого сравнения возможны три результата: Истина (True), Ложь (False) или Неизвестно (Unknown).

<i>AND</i>	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

<i>OR</i>	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

<i>NOT</i>	True	False	Unknown
	False	True	Unknown

NULL-значения в выражениях.

Функции, специально предназначенные для работы с неопределенными значениями.

ISNULL (<проверяемое поле>, < значение, если проверяемое поле равно NULL >)

преобразует NULL-значение к значению, отличному от NULL.

Для выявления равенства значения некоторого столбца неопределенному, применяют специальные стандартные предикаты;

<Столбец> IS NULL и **< Столбец > IS NOT NULL**.

NULL-значения в выражениях.

```
SELECT    Название,  
            ISNULL(Жанр, 'Не указан') as [Жанр книги]  
FROM      Книги  
WHERE     Жанр IS NULL OR Жанр = 'Детектив'
```

Использование BETWEEN

BETWEEN ... AND ... (находится в интервале от ... до ...) можно отобразить строки, в которых значение какого-либо столбца находятся в заданном диапазоне.

```
SELECT      *  
FROM        Книги  
WHERE       Тираж BETWEEN 10 000 AND 100 000
```

Использование IN (NOT IN).

Задаёт поиск выражения, включённого или исключённого из списка. Выражение поиска может быть константой или именем столбца, а списком может быть набор констант или, что чаще, вложенный запрос.

Список значений необходимо заключать в скобки.

```
SELECT   ФИО, Должность, Телефон  
FROM     Сотрудники  
WHERE    Должность IN ( 'редактор' , 'менеджер' )
```

```
SELECT   ФИО, Должность, Телефон  
FROM     Сотрудники  
WHERE    Номер IN ( SELECT [номер ответственного  
редактора]  
FROM Книги  
WHERE [Дата выхода] > '01.01.2016' )
```

Использование LIKE.

LIKE определяет, совпадает ли указанная символьная строка с заданным шаблоном.

выражение **[NOT] LIKE** строка_шаблон **[ESCAPE** esc_символ]

Символы строки_шаблона интерпретируются следующим образом:

символ **_** – заменяет любой одиночный символ,

символ **%** – заменяет любую последовательность из N

символов (где N может быть нулем),

все другие символы означают просто сами себя.

[] – любой одиночный символ внутри диапазона(**[a-f]**) или набора **[abcdef]**.

[^]- любой одиночный символ,

не принадлежащий диапазону (**[^a-f]**) или набору **[^abcdef]**.

Использование LIKE.

```
SELECT   ФИО, Должность, Телефон  
FROM     Сотрудники  
WHERE    Должность LIKE '% редактор'  
          OR Должность = 'менеджер'
```

Для проверки значения на соответствие строке «25%» можно воспользоваться таким предикатом:

```
.LIKE '25|%' ESCAPE '|'
```

```
.LIKE '[0-9][0-9]|%' ESCAPE '|'
```

Предикат EXISTS

[NOT] EXISTS (<табличный подзапрос>)

Предикат **EXISTS** принимает значение **TRUE**, если подзапрос содержит любое количество строк, иначе его значение равно **FALSE**. Для **NOT EXISTS** все наоборот. Этот предикат никогда не принимает значение **UNKNOWN**.

Обычно предикат **EXISTS** используется в зависимых (коррелирующих, соотнесенных) подзапросах. Этот вид подзапроса имеет внешнюю ссылку, связанную со значением в основном запросе. Результат подзапроса может зависеть от этого значения и должен оцениваться отдельно для каждой строки запроса, в котором содержится данный подзапрос. Поэтому предикат **EXISTS** может иметь разные значения для разных строк основного запроса.

Использование ключевых слов SOME (ANY) и ALL с предикатами сравнения

***<выражение> <оператор сравнения> SOME | ANY
(<подзапрос>)***

SOME и **ANY** являются синонимами, то есть может использоваться любое из них. Результатом подзапроса является один столбец величин. Если хотя бы для одного значения V , получаемого из подзапроса, результат операции "*<значение выражения> <оператор сравнения> V* " равняется **TRUE**, то предикат **ANY** также равняется **TRUE**.

<выражение> <оператор сравнения> ALL (<подзапрос>)

Исполняется так же, как и **ANY**, однако значение предиката **ALL** будет истинным, если для всех значений V , получаемых из подзапроса, предикат "*<значение выражения> <оператор сравнения> V* " дает **TRUE**.

Использование агрегатных функций для подведения итогов.

В SQL существует ряд специальных агрегатных (статических) функций.

COUNT(столбец) – возвращает количество строк с непустым значением (не NULL) в заданном столбце,

COUNT(*) – возвращает общее количество строк в выборке, включая строки со значением NULL,

SUM (столбец) – возвращает сумму всех значений в пределах группы в заданном столбце, применима только к столбцам с числовыми значениями,

AVG (столбец) – возвращает среднее арифметическое для указанного столбца в пределах строк, принадлежащих одной группе, применима только к столбцам с числовым типом данных,

MAX(столбец) - возвращает наибольшее значение в указанном столбце в пределах группы,

MIN (столбец) - возвращает наименьшее значение в

Использование агрегатных функций для подведения итогов.

Выражение, определяющее столбец такой таблицы, может быть сколь угодно сложным, но не должно содержать агрегатные функции (**вложенность агрегатных функций не допускается**). Однако из агрегатных функций можно составлять любые выражения.

Аргументу всех функций, кроме COUNT(*), может предшествовать ключевое слово ***DISTINCT*** (различный), указывающее, что избыточные дублирующие значения должны быть исключены перед тем, как будет применяться функция.

Агрегатные функции могут быть использованы в качестве выражений только в следующих случаях.

- Список выбора инструкции SELECT (вложенный или внешний запрос).
- Предложение HAVING.

Агрегатные функции без использования фразы GROUP BY.

Если не используется фраза **GROUP BY**, то в перечень элементов **_SELECT** можно включать лишь агрегатные функции или выражения, содержащие такие функции. Другими словами, нельзя иметь в списке столбцы, не являющихся аргументами агрегатных функций. Группой будет считаться вся выборка.

```
SELECT count(*) as [Количество клиентов] FROM КЛИЕНТЫ
```

```
SELECT max(Тираж) as [Наибольший тираж] FROM Книги
```

```
SELECT [номер ответственного редактора] FROM Книги  
WHERE тираж = (SELECT max(Тираж) as [Наибольший тираж]  
FROM Книги)
```

Фраза GROUP BY

Фраза **GROUP BY** (группировать по) инициирует перекомпоновку указанной во FROM таблицы по группам, каждая из которых имеет одинаковые значения в столбце, указанном в GROUP BY.

```
SELECT [Номер заказа],  
       count(*) as [Количество позиций] ,  
       sum(количество) as [Количество книг]  
FROM [Состав заказа]  
GROUP BY [Номер заказа]
```

Раздел HAVING.

Предложение **HAVING** подобно предложению WHERE, но применимо только к **целым группам** (то есть к строкам в результирующем наборе, представляющим собой группы), тогда как предложение **WHERE** применимо к **отдельным строкам**.

В запросе могут содержаться оба предложения: **WHERE** и **HAVING**. В этом случае:

1. Предложение **WHERE** применяется сначала к отдельным строкам таблиц или возвращающих табличное значение объектов в области схем. Группируются только строки, которые удовлетворяют условиям в предложении WHERE.
2. Затем предложение **HAVING** применяется к строкам в результирующем наборе. Только строки, которые удовлетворяют условиям HAVING, появляются в результирующем запросе. Можно применить предложение HAVING только к тем столбцам, которые появляются в предложении GROUP BY или статистической функции.

[**HAVING** условие_отбора_групп]

Раздел HAVING.

```
SELECT [Номер заказа],  
       count(*) as [Количество позиций] ,  
       sum(количество) as [Количество книг]  
  
FROM [Состав заказа]  
WHERE ISBN IN ( SELECT ISBN  
                FROM Книги  
                WHERE [Дата выхода]> GETDATE() )  
GROUP BY [Номер заказа]  
HAVING count(*) >10
```

Раздел HAVING.

R1(ФИО, Дисциплина, Оценка)

R2(ФИО, Группа)

R3(Группа, Дисциплина)

Найти студентов, имеющих лучший средний балл в своей группе.

Раздел HAVING.

```
SELECT [Номер заказа], count(*) as [Количество
книг]
FROM [Состав заказа]
WHERE ISBN IN ( SELECT ISBN
                FROM Книги
                WHERE [Дата выхода]> GETDATE() )
GROUP BY [Номер заказа]
HAVING
count(*) = ( SELECT max([Количество книг] )
             FROM (SELECT count(*) as [Количество книг]
                   FROM [Состав заказа]
                   WHERE ISBN IN (SELECT ISBN
                                 FROM Книги
                                 WHERE [Дата выхода]> GETDATE() )
```

Обобщенные табличные выражения (СТЕ).

Обобщенные табличные выражения (СТЕ) помогают повысить удобочитаемость (и, таким образом, возможность обслуживания) кода, не ухудшая производительности.

```
WITH [ RECURSIVE ] <имя_запроса> [ ( <список столбцов> ) ]  
AS ( <запрос select> )  
{, <имя_запроса> [ ( <список столбцов> ) ] AS ( <запрос  
select> ) }
```

<запрос, использующий имя_запроса>;

Выражения СТЕ могут оказаться полезными, когда запросам необходимо делать выборку из набора данных, не представленного в виде таблицы в БД.


```
select * from test_table
```

100 %

Результаты Сообщения

	id	manager	otdel	god	summa
1	1	Сотрудник_1	Бухгалтерия	2014	200,00
2	2	Сотрудник_2	Бухгалтерия	2014	300,00
3	3	Сотрудник_3	Отдел покупок	2014	150,00
4	4	Сотрудник_4	Отдел покупок	2014	200,00
5	5	Сотрудник_5	Отдел реализации	2014	250,00
6	6	Сотрудник_6	Отдел реализации	2014	300,00
7	7	Сотрудник_7	Отдел реализации	2014	300,00
8	8	Сотрудник_1	Бухгалтерия	2015	230,00
9	9	Сотрудник_2	Бухгалтерия	2015	200,00
10	10	Сотрудник_3	Отдел покупок	2015	200,00
11	11	Сотрудник_4	Отдел покупок	2015	300,00
12	12	Сотрудник_5	Отдел реализации	2015	200,00
13	13	Сотрудник_6	Отдел реализации	2015	250,00
14	14	Сотрудник_7	Отдел реализации	2015	350,00

```
select otdel, god, SUM(summa) as itog  
from dbo.test_table  
group by otdel, god  
order by otdel, god
```

100 %

Результаты Сообщения

	otdel	god	itog
1	Бухгалтерия	2014	500,00
2	Бухгалтерия	2015	430,00
3	Отдел покупок	2014	350,00
4	Отдел покупок	2015	500,00
5	Отдел реализации	2014	850,00
6	Отдел реализации	2015	800,00

Синтаксис фразы GROUP BY

GROUP BY [ALL] [CUBE | ROLLUP] {[таблица.]столбец [, [таблица.]столбец] ...}

ALL – означает включение в результат выборки всех групп, независимо от того, соответствуют ли связанные с ним данные существующим в разделе WHERE условиям выборки. В строках не соответствующих условию выборки, во всех столбцах, кроме столбцов, по которым осуществляется группировка, будут выведены значения NULL.

ROLLUP ()

Формирует статистические строки простого предложения GROUP BY и строки подытогов или строки со статистическими вычислениями высокого уровня, а также строки общего итога.

<i>SELECT</i>	<i>a, b, c, SUM(<expression>)</i>	<i>SELECT</i>	<i>a, b, c, SUM(<expression>)</i>
<i>FROM</i>	<i>T</i>	<i>FROM</i>	<i>T</i>
<i>GROUP BY</i>	<i>a, b, c</i>	<i>GROUP BY</i>	<i>ROLLUP(a, b, c)</i>

CUBE ()

Формирует статистические строки простого предложения GROUP BY, строки со статистическими вычислениями высокого уровня конструкции ROLLUP и строки с результатами перекрестных вычислений.

ROLLUP – оператор, который формирует промежуточные итоги для каждого указанного элемента и общий итог.

```
select otdel, god, SUM(summa) as itog
      from dbo.test_table
group by
rollup (otdel,god)
```

100 %

Результаты Сообщения

	otdel	god	itog
1	Бухгалтерия	2014	500,00
2	Бухгалтерия	2015	430,00
3	Бухгалтерия	NULL	930,00
4	Отдел покупок	2014	350,00
5	Отдел покупок	2015	500,00
6	Отдел покупок	NULL	850,00
7	Отдел реализации	2014	850,00
8	Отдел реализации	2015	800,00
9	Отдел реализации	NULL	1650,00
10	NULL	NULL	3430,00

```
select otdel, SUM(summa) as itog
      from dbo.test_table
group by
rollup (otdel)
```

100 %

Результаты Сообщения

	otdel	itog
1	Бухгалтерия	930,00
2	Отдел покупок	850,00
3	Отдел реализации	1650,00
4	NULL	3430,00

```
select god, SUM(summa) as itog
      from dbo.test_table
group by
rollup (god)
```

100 %

Результаты Сообщения

	god	itog
1	2014	1700,00
2	2015	1730,00
3	NULL	3430,00

CUBE — оператор , который формирует результаты для всех возможных перекрестных вычислений.

```
select otdel, god, SUM(summa) as itog
  from dbo.test_table
 group by
  cube (otdel, god)
```

100 %

Результаты Сообщения

	otdel	god	itog
1	Бухгалтерия	2014	500,00
2	Отдел покупок	2014	350,00
3	Отдел реализации	2014	850,00
4	NULL	2014	1700,00
5	Бухгалтерия	2015	430,00
6	Отдел покупок	2015	500,00
7	Отдел реализации	2015	800,00
8	NULL	2015	1730,00
9	NULL	NULL	3430,00
10	Бухгалтерия	NULL	930,00
11	Отдел покупок	NULL	850,00
12	Отдел реализации	NULL	1650,00

GROUPING SETS – оператор, который формирует результаты нескольких группировок в один набор данных, другими словами, он эквивалентен конструкции UNION ALL к указанным группам.

```
select otdel, god, SUM(summa) as itog
  from dbo.test_table
 group by
  grouping sets (otdel,god)
```

100 %

Результаты Сообщения

	otdel	god	itog
1	NULL	2014	1700,00
2	NULL	2015	1730,00
3	Бухгалтерия	NULL	930,00
4	Отдел покупок	NULL	850,00
5	Отдел реализации	NULL	1650,00

```
select null as otdel, god, SUM(summa) as itog
  from dbo.test_table
 group by god
 union all
select otdel, null as god, SUM(summa) as itog
  from dbo.test_table
 group by otdel
```

100 %

Результаты Сообщения

	otdel	god	itog
1	NULL	2014	1700,00
2	NULL	2015	1730,00
3	Бухгалтерия	NULL	930,00
4	Отдел покупок	NULL	850,00
5	Отдел реализации	NULL	1650,00

GROUPING – функция Transact-SQL, которая возвращает истину, если указанное выражение является статистическим, и ложь, если выражение нестатистическое.

Данная функция создана для того, чтобы отличить статистические строки, которые добавил SQL сервер, от строк, которые и есть сами данные, так как

КО.

```
select otdel,
       ISNULL(cast(god as varchar(30)),
              case when GROUPING(god)=1 and GROUPING(otdel)=0
                   then 'Промежуточный итог' else 'Общий итог' end) as god,
       SUM(summa) as itog,
       GROUPING(otdel) as grouping_otdel,
       GROUPING(god) as grouping_god
from dbo.test_table
group by
rollup (otdel,god)
```

100 %

Результаты Сообщения

	otdel	god	itog	grouping_otdel	grouping_god
1	Бухгалтерия	2014	500,00	0	0
2	Бухгалтерия	2015	430,00	0	0
3	Бухгалтерия	Промежуточный итог	930,00	0	1
4	Отдел покупок	2014	350,00	0	0
5	Отдел покупок	2015	500,00	0	0
6	Отдел покупок	Промежуточный итог	850,00	0	1
7	Отдел реализации	2014	850,00	0	0
8	Отдел реализации	2015	800,00	0	0
9	Отдел реализации	Промежуточный итог	1650,00	0	1
10	NULL	Общий итог	3430,00	1	1

Выражение CASE

Оценка списка условий и возвращение одного из нескольких возможных выражений результатов.

Выражение CASE имеет два формата:

- простое выражение CASE для определения результата сравнивает выражение с набором простых выражений;
- поисковое выражение CASE для определения результата вычисляет набор логических выражений.

Оба формата поддерживают дополнительный аргумент ELSE.

Выражение CASE может использоваться в любой инструкции или предложении, которые допускают использование выражения данного типа.

--Simple CASE expression:

```
CASE input_expression  
    WHEN when_expression THEN result_expression [ ...n ]  
    [ ELSE else_result_expression ]  
END
```

--Searched CASE expression:

```
CASE  
    WHEN Boolean_expression THEN result_expression [ ...n ]  
    [ ELSE else_result_expression ]  
END
```

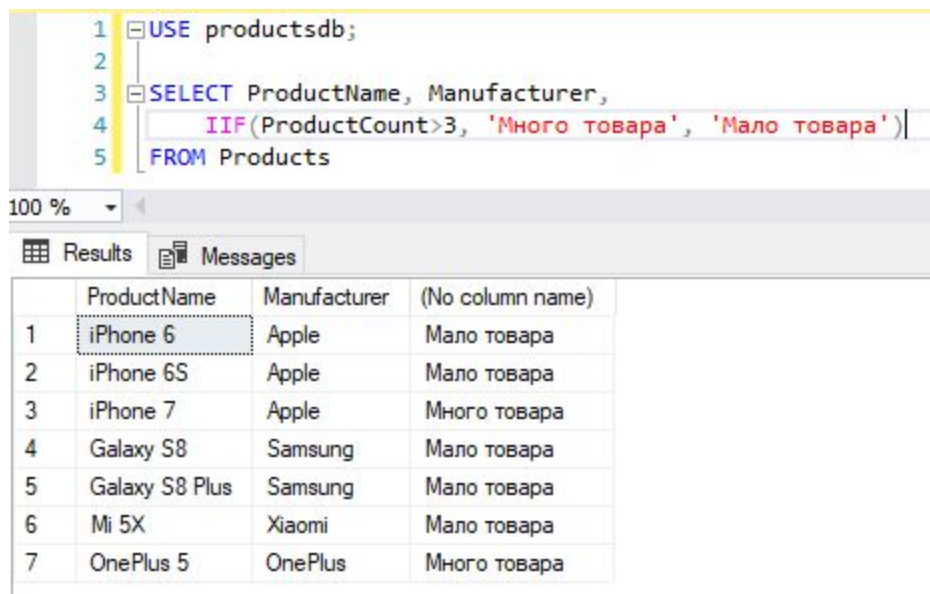
IIF

Функция IIF в зависимости от результата условного выражения возвращает одно из двух значений.

Общая форма функции выглядит следующим образом:

IIF(условие, значение_1, значение_2)

```
SELECT ProductName, Manufacturer,  
       IIF(ProductCount>3, 'Много товара', 'Мало товара')  
FROM Products
```



The screenshot shows a SQL query window with the following code:

```
1 USE productsdb;  
2  
3 SELECT ProductName, Manufacturer,  
4        IIF(ProductCount>3, 'Много товара', 'Мало товара')  
5 FROM Products
```

Below the query window, the 'Results' tab is active, displaying a table with the following data:

	ProductName	Manufacturer	(No column name)
1	iPhone 6	Apple	Мало товара
2	iPhone 6S	Apple	Мало товара
3	iPhone 7	Apple	Много товара
4	Galaxy S8	Samsung	Мало товара
5	Galaxy S8 Plus	Samsung	Мало товара
6	Mi 5X	Xiaomi	Мало товара
7	OnePlus 5	OnePlus	Много товара

Соединения «с условием WHERE».

Соединения - это подмножества декартова произведения.

```
SELECT *  
FROM    Клиент, Заказ  
WHERE   Клиент.Номер = Заказ.[Номер Клиента]  
         and  
         Клиент.ФИО='Иванов'
```

Операторы соединения в SQL92

CROSS JOIN

NATURAL JOIN

SPECIFIED JOIN

UNION JOIN

ON

USING

INNER JOIN

OUTER JOIN

Продавцы(Npr, Namepr, city, comm)
Покупатели(Np, Namep, city, rating, Npr)
SELECT *
FROM Продавцы Join Покупатели Using(city)

LEFT OUTER JOIN
RIGHT OUTER JOIN
FULL OUTER JOIN

Соединения нескольких таблиц, используя JOIN.

Существует три основных типа соединения:

□ **внутреннее соединение**, задаваемое с помощью ключевых слов INNER JOIN

*FROM таблица_A [INNER] JOIN таблица_B
ON условие_соединения*

□ **внешнее соединение**, которое может принимать три формы:

LEFT OUTER JOIN

RIGHT OUTER JOIN

FULL OUTER JOIN

*FROM таблица_A { LEFT | RIGHT | FULL } [OUTER] JOIN
таблица_B
ON условие_соединения*

□ **перекрёстное соединение**, задаваемое ключевыми

Внутреннее соединение

Во **внутреннем** соединении возвращаются *только те строки,*
которые соответствуют условию, указанному после
ключевого слова ON.

```
SELECT   A, B  
FROM     R1 INNER JOIN R2 ON A=B;
```

A	B
102	101
104	102
106	104
107	106
	108

Результат	
A	B
102	102
104	104
106	106

Левое внешнее соединение.

В левом внешнем соединении *результатом являются все строки*

левой таблицы, вне зависимости от того, имеют ли они подходящую пару в правой таблице.

```
SELECT   A, B  
FROM     R1 LEFT OUTER JOIN R2 ON A=B;
```

A	B
102	101
104	102
106	104
107	106
	108

Результат	
A	B
102	102
104	104
106	106
107	NULL

Правое внешнее соединение.

В правом внешнем соединении *результатом являются все строки правой таблицы, вне зависимости от того, имеют ли они подходящее соответствие в левой таблице.*

```
SELECT    A, B  
FROM      R1 RIGHT OUTER JOIN R2 ON A=B;
```

<i>A</i>	<i>B</i>
102	101
104	102
106	104
107	106
	108

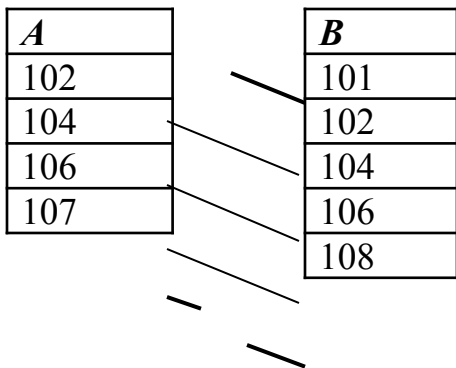
Результат	
<i>A</i>	<i>B</i>
<i>NULL</i>	<i>101</i>
102	102
104	104
106	106
<i>NULL</i>	<i>108</i>

Полное внешнее соединение.

В полном внешнем соединении *результатом являются строки*

обеих таблицы, вне зависимости от того, имеют ли они соответствия в другой таблице.

```
SELECT    A, B  
FROM      R1 FULL OUTER JOIN R2 ON A=B;
```



Результат	
<i>A</i>	<i>B</i>
<i>NULL</i>	<i>101</i>
102	102
104	104
106	106
<i>107</i>	<i>NULL</i>
<i>NULL</i>	<i>108</i>

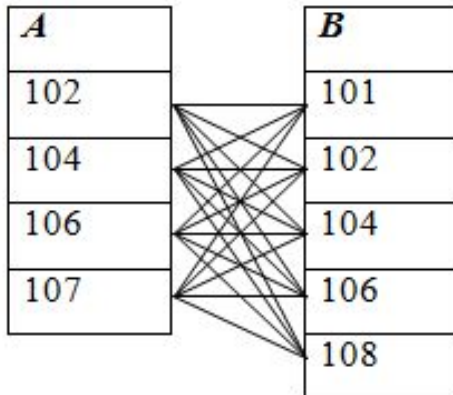
Перекрёстное соединение.

В перекрёстном соединении *каждая строка из одной таблицы*

соединяется с каждой строкой из другой таблицы.

Отличительной чертой перекрёстного соединения является отсутствие условия ON.

SELECT A, B FROM R1 CROSS JOIN R



Результат	
<i>A</i>	<i>B</i>
102	101
102	102
102	104
102	106
102	108
104	101
104	102
104	104
104	106
104	108

<u>Продолжение</u>	
106	101
106	102
106	104
106	106
106	108
107	101
107	102
107	104
107	106
107	108

□

PIVOT и UNPIVOT

```
SELECT maker, type FROM product;
```



Maker type

B PC
A PC
A PC
E PC
A Printer
D Printer
A Laptop
C Laptop
A Printer
A Printer
D Printer
E Printer
B Laptop
A Laptop
E PC
E PC

Типы продукции

П	Laptop	PC	Printer
р			
о	A 2	2	3
и			
з	B 1	1	0
в			
о	C 2	0	0
д			
и	D 0	0	2
т			
е	E 0	3	1
л			
и			

```
SELECT maker,  
       SUM(CASE type WHEN 'pc' THEN 1 ELSE 0 END) PC,  
       SUM(CASE type WHEN 'laptop' THEN 1 ELSE 0 END) Laptop,  
       SUM(CASE type WHEN 'printer' THEN 1 ELSE 0 END) Printer  
FROM Product  
GROUP BY maker
```

```
SELECT maker, [pc], [laptop], [printer]  
FROM Product  
PIVOT  
(COUNT(model)  
  FOR type IN ([pc], [laptop], [printer])  
) pvt
```

PIVOT и UNPIVOT

```
SELECT <non-pivoted column>,  
    [first pivoted column] AS <column name>,  
    [second pivoted column] AS <column name>,  
    ...  
    [last pivoted column] AS <column name>  
FROM  
    (<SELECT query that produces the data>  
    AS <alias for the source query>  
PIVOT  
(  
    <aggregation function>( <column being aggregated>)  
FOR  
[<column that contains the values that will become column headers>]  
    IN ( [first pivoted column], [second pivoted column],  
    ... [last pivoted column])  
) AS <alias for the pivot table>  
<optional ORDER BY clause>;
```

```
SELECT screen, AVG(price) avg_ FROM Laptop GROUP BY screen
```

<u>screen</u>	<u>avg_</u>
11	700.00
12	960.00
14	1175.00
15	1050.00

А вот как можно повернуть эту таблицу с помощью PIVOT:

```
SELECT [avg_], [11],[12],[14],[15]
      FROM (SELECT 'average price' AS 'avg_', screen, price FROM Laptop)
      PIVOT (AVG(price) FOR screen IN([11],[12],[14],[15])
            ) pvt
```

<u>avg_</u>	<u>11</u>	<u>12</u>	<u>14</u>	<u>15</u>
average price	700.00	960.00	1175.00	1050.00

<u>trip_no</u>	<u>id_comp</u>	<u>plane</u>	<u>town_from</u>	<u>town_to</u>	<u>time_out</u>	<u>time_in</u>
1100	4	Boeing	Rostov	Paris	14:30:00	17:50:00

<u>trip_no</u>	<u>spec</u>	<u>info</u>
1100	id_comp	4
1100	Plane	Boeing
1100	town_from	Rostov
1100	town_to	Paris
1100	time_out	14:30:00
1100	time_in	17:50:00

```

SELECT trip_no, spec, info
FROM ( SELECT trip_no,
      CAST(id_comp AS CHAR(25)) id_comp,
      CAST(plane AS CHAR(25)) plane,
      CAST(town_from AS CHAR(25)) town_from,
      CAST(town_to AS CHAR(25)) town_to,
      CONVERT(CHAR(25),time_out, 108) time_out,
      CONVERT(CHAR(25),time_in,108) time_in
      FROM Trip WHERE trip_no =1100 )
UNPIVOT( info FOR spec IN (id_comp, plane, town_from, town_to, time_out,
time_in)
) unpvt;

```

```

select [Вид],[Овощи],[Мясо],[Рыба],[Молоко],[Яйца],[Крупы]
from (Select b.[Вид],[Основа]
      from [dbo].[Блюда] a join [dbo].[Справочник_вид_блюда] b
      on a.[Вид]=b.[Id_вид]) as a
pivot(count([Основа]) for [Основа] in([Овощи],[Мясо],[Рыба],[Молоко],
[Яйца],[Крупы])) pvt1

```

Вид	Овощи	Мясо	Рыба	Молоко	Яйца	Крупа	Фрукты	Кофе
Горячее	2	2	1	3	2	2	0	0

```

Select Основа, Горячее
from(select [Вид],[Овощи],[Мясо],[Рыба],[Молоко],[Яйца],[Крупы]
      from (Select b.[Вид],[Основа]
            from [dbo].[Блюда] a join
            [dbo].[Справочник_вид_блюда] b
            on a.[Вид]=b.[Id_вид]) as a
      pivot(count([Основа]) for [Основа] in ([Овощи],[Мясо],[Рыба],
            [Молоко],[Яйца],[Крупы])) pvt1
      where [Вид]='Горячее') pvt2
unpivot(Горячее for Основа
in ([Овощи],[Мясо],[Рыба],[Молоко],[Яйца],[Крупы])
) unpvt

```

Основа	Горячее
Овощи	2
Мясо	2
Рыба	1
Молоко	3
Яйца	2
Крупа	2
Фрукты	0
Кофе	0