Базы данных и SQL

Лекция 19

Адукар

Повторим изученное на прошлом занятии

Что такое первичный и ссылочный ключ?
Что такое Реляционная база данных?
Что такое СУБД?
Как будет выглядеть запрос, если мы ищем что-то с конкретным условием?
Оператор для интервального запроса? (between)
Как оператор исключает повторы?
Что используем, если помнишь только несколько букв из фамилии?

Адукар

Что будем изучать сегодня

Связывание сущностей 1:1 и 1: многим
Обеспечение целостности
Джоины
Объединение (union)
Minus и Except
Группировки
SUM и AVG и Having

В этой презентации мы попробуем понять, как связывать таблицы друг с другом, но изначально необходимо определиться с нюансами и тонкостями этих связок. Связь позволяет моделировать отношения между объектами предметной области. Существует 3 типа связей:

1. «Один-к-одному» (1:1) - любому экземпляру сущности A соответствует только один экземпляр сущности B, и наоборот.

Например (обыкновенно), у одного мужа может быть лишь одна жена и у одной жены – только один муж.

2. «Один-ко-многим» (1:M) - любому экземпляру сущности A соответствует 0, 1 или несколько экземпляров сущности B, но любому экземпляру сущности B соответствует только один экземпляр сущности A

Например, у поэта может быть множество стихов, но автор у них один.

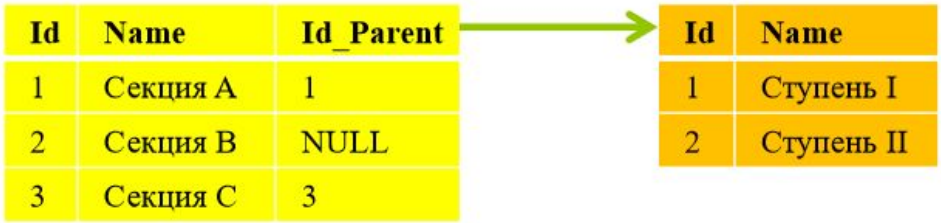
АДУКАР

3. «Многие-ко-многим» - любому экземпляру сущности A соответствует 0, 1 или несколько экземпляров сущности B, и любому экземпляру сущности B соответствует 0, 1 или несколько экземпляров сущности A.

Например, у каждого преподавателя в университете может быть множество студентов. В свою очередь, каждого студента обучает некоторое множество преподавателей.

Обеспечение целостности

В теории баз данных целостность данных означает корректность данных и их непротиворечивость. Обычно, она также включает целостность связей, которая исключает ошибки связей между первичным и вторичным ключом. К примеру, когда существуют дочерние записи-сироты, которые не имеют связи с родительскими записями. В данном случае, таблица с секциями ссылается на таблицу со ступенями (M:1), но у секции B отсутствует ссылка на родительскую запись, а секция C вообще ссылается на несуществующую ступень.



Id	Name	Id_Parent
1	Секция A	1
2	Секция B	NULL
3	Секция C	3

Id	Name
1	Ступень I
2	Ступень II

Чтобы обеспечить целостность, работа с данными должна производиться с учетом нижеперечисленных правил:

1. Невозможно ввести в связанное поле подчиненной таблицы значение, отсутствующее в связанном поле главной таблицы. Однако можно ввести пустое значение, показывающее, что для данной записи связь отсутствует.
2. Не допускается удаление записи из главной таблицы, если существуют связанные с ней записи в подчиненной таблице.
3. Невозможно изменить значение ключевого поля в главной таблице, если существуют записи, связанные с данной таблицей.

JOIN

Переходим непосредственно к коду. Наша задача – научиться связывать несколько таблиц в рамках одного запроса. В этом нам поможет оператор JOIN. Ключевое слово JOIN в SQL используется при построении select-выражений. Инструкция JOIN позволяет объединить колонки из нескольких таблиц в одну. Объединение происходит временное и целостность таблиц не нарушается. Существует три типа join-выражений: inner join; outer join; cross join; В свою очередь, outer join может быть left, right и full (слова inner и outer обычно опускаются). В качестве примера возьмём две простые таблицы (таблица A с работниками ссылается на таблицу B с должностями) и сконструируем для них SQL-выражения с использованием join.

id	name	Id_B
1	Гришин	1
2	Васильев	2
3	Петров	3
4	Иванов	NULL

id	name
1	Бухгалтерия
2	IT
3	Снабжение
4	Охрана

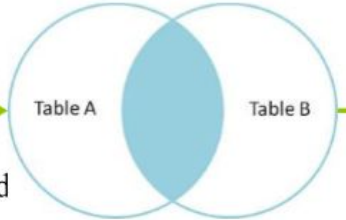
АДУКАР

Inner Join

id	name	Id_B
1	Гришин	1
2	Васильев	2
3	Петров	3
4	Иванов	NULL

id	name
1	Бухгалтерия
2	IT
3	Снабжение
4	Охрана

```
SELECT *
FROM a
JOIN b
ON a.id_B = b.id
```



id	name	Id_B	Id	name
1	Гришин	1	1	Бухгалтерия
2	Васильев	2	2	IT
3	Петров	3	3	Снабжение

Обратите внимание на диаграмму множеств. Внутреннее соединение INNER JOIN производит выборку только тех записей, которые соответствуют пересечению таблиц A и B. Работник «Иванов» из таблицы A не нашёл себе соответствий в таблице B, также, как и вид должности «Охрана» из таблицы B не привязан ни к одному работнику, поэтому данные значения не попали в конечный результат выборки.

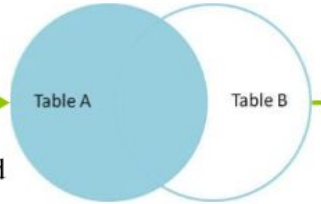
АДУКАР

LEFT OUTER JOIN

id	name	Id_B
1	Гришин	1
2	Васильев	2
3	Петров	3
4	Иванов	NULL

id	name
1	Бухгалтерия
2	IT
3	Снабжение
4	Охрана

```
SELECT *
FROM a
LEFT JOIN b
ON a.id_B = b.id
```



id	name	Id_B	Id	name
1	Гришин	1	1	Бухгалтерия
2	Васильев	2	2	IT
3	Петров	3	3	Снабжение
4	Иванов	NULL	NULL	NULL

Левостороннее внешнее соединение LEFT OUTER JOIN производит полный выбор записей из таблицы, содержащейся во FROM (в данном случае A) с соответствующими записями (если таковые имеются) таблицы B. Если совпадения нет, то правая часть будет содержать NULL

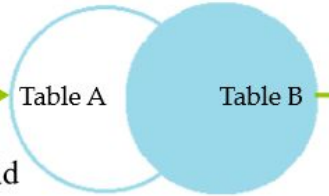
АДУКАР

RIGHT OUTER JOIN

id	name	Id_B
1	Гришин	1
2	Васильев	2
3	Петров	3
4	Иванов	NULL

id	name
1	Бухгалтерия
2	IT
3	Снабжение
4	Охрана

```
SELECT *
FROM a
RIGHT JOIN b
ON a.id_B = b.id
```



id	name	Id_B	Id	name
1	Гришин	1	1	Бухгалтерия
2	Васильев	2	2	IT
3	Петров	3	3	Снабжение
NULL	NULL	NULL	4	Охрана

Правостороннее внешнее соединение действует по аналогии с LEFT JOIN'ом. RIGHT OUTER JOIN производит полный выбор записей из таблицы, которая указана после слов RIGHT JOIN (в данном случае из B) с соответствующими записями (если таковые имеются) таблицы A. Если совпадения нет, то левая часть будет содержать NULL.

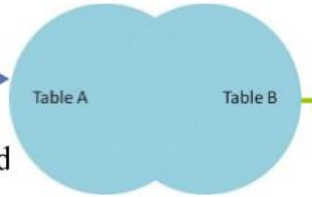
АДУКАР

FULL OUTER JOIN

id	name	Id_B
1	Гришин	1
2	Васильев	2
3	Петров	3
4	Иванов	NULL

id	name
1	Бухгалтерия
2	IT
3	Снабжение
4	Охрана

```
SELECT *
FROM a
FULL JOIN b
ON a.id_B = b.id
```



id	name	Id_B	Id	name
1	Гришин	1	1	Бухгалтерия
2	Васильев	2	2	IT
3	Петров	3	3	Снабжение
4	Иванов	NULL	NULL	NULL
NULL	NULL	NULL	4	Охрана

Полное внешнее соединение FULL OUTER JOIN производит выборку множества всех записей из таблицы A и B с соответствующими записями с обеих сторон при их наличии. Если совпадения нет, отсутствующая сторона будет содержать NULL. Обратите внимание на две последних строки в итоговой выборке.

АДУКАР

CROSS JOIN

Также существует выборка перекрестного соединения (называемое ещё декартовым произведением), - CROSS JOIN, с перебором всех вариантов, которое сложно объяснить диаграммами:

```
SELECT *
FROM a CROSS JOIN b;
 ---(ещё можно написать запрос как ---SELECT * FROM a, b)
```

Данное перекрестное соединение выбирает буквально "всё ко всему", в результате мы получим 4 x 4 = 16 записей, т.е. намного больше, чем в оригинале мы имеем в таблицах

id	name	Id_B	Id	name
1	Гришин	1	1	Бухгалтерия
1	Гришин	1	2	IT
1	Гришин	1	3	Снабжение
1	Гришин	1	4	Охрана
2	Васильев	2	1	Бухгалтерия
2	Васильев	2	2	IT
2	Васильев	2	3	Снабжение
2	Васильев	2	4	Охрана
3	Петров	3	1	Бухгалтерия
3	Петров	3	2	IT
3	Петров	3	3	Снабжение
3	Петров	3	4	Охрана
4	Иванов	NULL	1	Бухгалтерия
4	Иванов	NULL	2	IT
4	Иванов	NULL	3	Снабжение
4	Иванов	NULL	4	Охрана

АДУКАР

Работа со множествами

Прежде чем начать описание функций и их свойств в SQL, работающих со множествами – дадим определение термину “множество” на языке SQL. Ведь многим из нас известно, что множество - это один из ключевых объектов математики, в частности, теории множеств и логики. По определению Бертрана Рассела: “Множество – совокупность различных элементов, мыслимая как единое целое” .

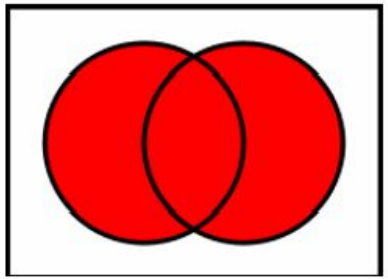
А сейчас переместим это определение в базу данных и увидим, что вывод любого запроса можно принять за множество. Наверное у вас возникнет вопрос в виде: “И что же это нам дает?” Ответ прост – применение некоторых свойств и операций, присущих множествам. Сейчас мы с вами разберем, что это за операции и как они реализованы в SQL.

АДУКАР

Объединение. UNION

Объединением двух множеств A и B называется множество, содержащее в себе все элементы исходных множеств, и обозначается как A ∪ B. В языке SQL ключевое слово UNION применяется для объединения результатов двух SQL-запросов в единую таблицу, состоящую из схожих строк. Оба запроса должны возвращать одинаковое число столбцов и совместимые типы данных в соответствующих столбцах!!! Оператор указывается между запросами. В упрощенном виде это выглядит следующим образом:

<запрос1> UNION [ALL]
<запрос2> UNION [ALL]
<запрос3> .....;



По умолчанию, любые дублирующие записи автоматически скрываются(своего рода default-ный DISTINCT), если не использовано выражение UNION ALL. Необходимо отметить, что UNION сам по себе не гарантирует порядок строк. Строки из второго запроса могут оказаться в начале, в конце или вообще перемешаться со строками из первого запроса. В случаях, когда требуется определенный порядок, необходимо использовать выражение ORDER BY

АДУКАР

Как выглядит объединение множеств на языке SQL:

sales2005

person	amount
Иван	1000
Алексей	2000
Сергей	5000

U

sales2006

person	amount
Иван	2000
Алексей	2000
Петр	35000

=

person	amount
Иван	1000
Алексей	**2000**
Иван	2000
Сергей	5000
Петр	35000

(SELECT * FROM sales2005)
UNION (SELECT * FROM sales2006);

В результате отобразятся две строки с Иваном, так как эти строки различаются значениями в столбцах. Но при этом в результате присутствует лишь одна строка с Алексеем, поскольку значения в столбцах полностью совпадают. Применение UNION ALL дает другой результат, так как дубликаты не скрываются. Выполнение запроса:

(SELECT * FROM sales2005)
UNION ALL
(SELECT * FROM sales2006);

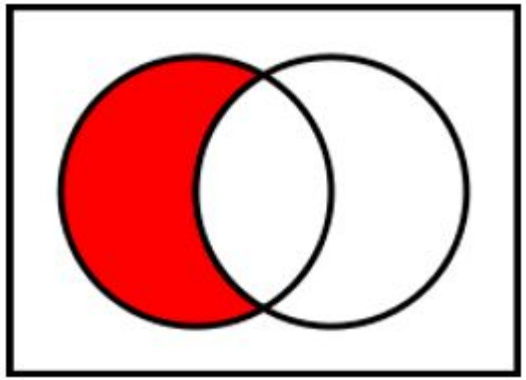
даст следующий результат:

person	amount
Иван	1000
Иван	2000
Алексей	**2000**
Алексей	**2000**
Сергей	5000
Петр	35000

АДУКАР

Разность. MINUS или EXCEPT

Разность двух множеств — это операция, результатом которой является множество, в которое входят все элементы первого множества, не входящие во второе множество. A\B означает множество элементов, принадлежащих A, но не принадлежащих B. В языке SQL нет стандарта для оператора разности множеств, поэтому он может в некоторых СУБД принимать значение MINUS, в других – EXCEPT. В любом случае, как и UNION, оператор разности обладает ограничением на результат запросов, которые должны быть совместимы по объединению, т.е. содержать одинаковое количество столбцов, и каждый столбец первого запроса должен быть того же типа данных (или автоматически приводиться к нему), что и находящийся в том же месте столбец второго запроса.



<запрос1> MINUS/EXCEPT
<запрос2> MINUS/EXCEPT
<запрос3> .....;

Как выглядит разность множеств на языке SQL (версия для СУБД MS SQL Server):

sales2005

person	amount
Иван	1000
Алексей	2000
Сергей	5000

\

sales2006

person	amount
Иван	2000
Алексей	2000
Петр	35000

=

person
Сергей

(SELECT person FROM sales2005) EXCEPT (SELECT person FROM sales2006); В результате отобразится строка с Сергеем, уникальным именем для sales2005, т.к. все остальные имена были исключены множеством имён sales2006. Обратите внимание, что Пётр в конечную выборку не попал.

Агрегаты

Очень часто в языке SQL ставятся задачи выделения по набору данных максимальных, минимальных или иных значений поля. Теми средствами, которые были нами изучены ранее, получить эти значения проблематично, поэтому на помощь приходят агрегатные функции. Агрегатные функции выполняют вычисление на наборе значений и возвращают одиночное значение.

Агрегаты, за исключением COUNT, не учитывают значения NULL. Эти функции часто используются в выражении GROUP BY инструкции SELECT.

Все агрегатные функции являются детерминированными. Это означает, что агрегатные функции возвращают одну и ту же величину при каждом их вызове на одном и том же наборе входных значений.

Агрегаты

Список функций, который входит в стандарт SQL:

COUNT – функция возвращает количество элементов в группе или количество строк в таблице.
SUM – возвращает арифметическую сумму всех выбранных значений данного поля.
AVG – возвращает среднее всех выбранных значений данного поля.
MAX – возвращает наибольшее из всех выбранных значений данного поля.
MIN – возвращает наименьшее из всех выбранных значений данного поля.

Адукар

Группировка

Выражение GROUP BY используется для определения групп выходных строк, к которым могут применяться агрегатные функции (COUNT, MIN, MAX, AVG и SUM). Если это предложение отсутствует, и используются агрегатные функции, то все столбцы с именами, упомянутыми в SELECT, должны быть включены в агрегатные функции, и эти функции будут применяться ко всему набору строк, которые удовлетворяют предикату запроса.

В противном случае все столбцы списка SELECT, не вошедшие в агрегатные функции, должны быть указаны в предложении GROUP BY.

В результате чего все выходные строки запроса разбиваются на группы, характеризуемые одинаковыми комбинациями значений в этих столбцах. После чего к каждой группе будут применены агрегатные функции.

Группировка

Следует иметь в виду, что для GROUP BY все значения NULL трактуются как равные, то есть при группировке по полю, содержащему NULL-значения, все такие строки попадут в одну группу.
Если при наличии предложения GROUP BY, в предложении SELECT отсутствуют агрегатные функции, то запрос просто вернет по одной строке из каждой группы. Эту возможность, наряду с ключевым словом DISTINCT, можно использовать для исключения дубликатов строк в результирующем наборе.

Адукар

Функция COUNT(*)

Функция COUNT(*) возвращает количество элементов в группе. Сюда входят NULL и повторяющиеся значения. Например:

```
SELECT COUNT(*)
FROM sales2005
```

вернёт 5, поскольку строк в таблице sales2005 пять штук. Функция COUNT(название_поля) оценивает поле построчно и возвращает количество значений, не равных NULL.

Например:

```
SELECT COUNT(amount) FROM sales2005
```

вернёт 4, поскольку в таблице sales2005 в поле amount содержится четыре строки, значения которых не равны NULL.

sales2005

person	amount
Иван	NULL
Алексей	2000
Иван	5000
Алексей	3000
Пётр	4000

АДУКАР

Функция COUNT(*)

Функция COUNT(DISTINCT имя_поля) оценивает значения в поле для каждой строки в группе и возвращает количество уникальных значений, не равных NULL.

Например:

```
SELECT COUNT(DISTINCT person)
FROM sales2005
```

вернёт 3, поскольку в таблице sales2005 в поле person содержится три уникальных имени, не смотря на то, что всего значений там пять. Как вы считаете, что произойдёт, если добавить группировку по полю person?

sales2005

person	amount
Иван	NULL
Алексей	2000
Иван	5000
Алексей	3000
Пётр	4000

АДУКАР

SUM

Возвращает сумму всех, либо только уникальных (при наличии DISTINCT), значений в выражении. Функция SUM может быть использована только для числовых столбцов. Псевдозначения NULL пропускаются.

Например:
SELECT SUM(amount)
FROM sales2005
вернёт сумму всех значений поля amount, т.е. 14000.

Теперь попробуем добавить группировку по имени:

SELECT person, SUM(amount)
FROM sales2005
GROUP BY person
Наш запрос вернёт следующий результат:

person	amount
Алексей	5000
Иван	5000
Пётр	4000

sales2005

person	amount
Иван	NULL
Алексей	2000
Иван	5000
Алексей	3000
Пётр	4000

АДУКАР

AVG

Возвращает среднее арифметическое группы значений. Пустые множества NULL пропускаются. Функция AVG() вычисляет среднее арифметическое набора значений, выполняя деление суммы этих значений на количество значений, не равных NULL. Например:

```
SELECT  AVG(amount)
FROM sales2005
```

Запрос вернёт среднее всех значений поля amount, т.е. 3500.

Теперь попробуем добавить группировку по имени:

```
SELECT  person, AVG(amount)
FROM sales2005
GROUP BY person
```

Наш запрос вернёт следующий результат:

person	amount
Алексей	2500
Иван	5000
Пётр	4000

sales2005

person	amount
Иван	NULL
Алексей	2000
Иван	5000
Алексей	3000
Пётр	4000

АДУКАР

MAX() – возвращает максимальное значение выражения.
MIN() – возвращает минимальное значение выражения.
NULL пропускается для обеих функций.

При использовании со столбцами, содержащими символьные значения, функция MAX находит наибольшее значение в упорядоченной последовательности, а MIN - наименьшее.

Например:
SELECT MAX(person)
FROM sales2005
вернёт максимальное значение поля person (по алфавиту, но, если точнее, то по ASCII), т.е. «Пётр».

Теперь попробуем добавить группировку по имени и вычислить максимальные и минимальные цифры:
SELECT person, MAX(amount) AS 'Max', MIN(amount) AS 'Min'
FROM sales2005
GROUP BY person
Наш запрос вернёт следующий результат:

person	Max	Min
Алексей	3000	2000
Иван	5000	5000
Пётр	4000	4000

АДУКАР

Важный момент: большой плюс агрегатов состоит в том, что, в рамках одного запроса, вы можете вызвать их сколько угодно.

Например:

SELECT person, MAX(amount) AS mx, MIN(amount) AS mn, COUNT(amount) AS cnt, AVG(amount) AS avrg SUM(amount) AS sm FROM sales2005 GROUP BY person

Этот код вернёт следующий результат:

person	mx	mn	cnt	avrg	sm
Алексей	3000	2000	2	2500	5000
Иван	5000	5000	1	5000	5000
Пётр	4000	4000	1	4000	4000

Главное: не забывать группировать!

АДУКАР

HAVING — необязательный (опциональный) параметр оператора SELECT для указания условия на результат агрегатных функций (MAX, SUM, AVG, …). HAVING аналогичен WHERE за исключением того, что строки отбираются не по значениям столбцов, а строятся из значений столбцов, указанных в GROUP BY, и значений агрегатных функций, вычисленных для каждой группы, образованной GROUP BY . Иными словами, WHERE предназначен для фильтрации по полям, а HAVING – для фильтрации по группам
Необходимо, чтобы в SELECT были заданы только требуемые в выходном потоке столбцы, перечисленные в GROUP BY и/или агрегированные значения. Распространённая ошибка — указание в SELECT столбца, пропущенного в GROUP BY. Если параметр GROUP BY в SELECT не задан, HAVING применяется к «группе» всех строк таблицы, полностью дублируя WHERE (допускается не во всех реализациях стандарта SQL).
Например, мы хотим получить людей, у которых суммарное количество средств превышает 4 тысячи + вывести средний показатель дохода по таким людям:

```
SELECT person, AVG(amount) AS my_avg
FROM sales2005
GROUP BY person
HAVING SUM(amount) > 4000
```

person	my_avg
Алексей	2500
Иван	5000

АДУКАР

Исходя из изученных нами положений, выведем общий синтаксис запроса:

```
SELECT [DISTINCT] {список_вывода_через_запятую | *}
 FROM таблица_1 AS псевдоним_1 [INNER/LEFT/RIGHT/FULL] JOIN таблица_2 AS
псевдоним_2
ON псевдоним_1.поле_1 = псевдоним_2.поле_2
 [INNER/LEFT/RIGHT/FULL] JOIN …        --- сколько угодно JOIN'ов
 ON … AND …   --- возможны множественные условия
WHERE условие_отбора_записей     --- с AND, OR, IN и т.д.
 GROUP BY поля_через_запятую
 HAVING условие_отбора_групп  --- с AND, OR, IN и т.д.

UNION / EXCEPT / INTERSECT  --- если ведётся работа с множествами

SELECT ….    -- новый запрос (множество) аналогично
 …
 ORDER BY поля_через_запятую
```

Что изучили сегодня

Связывание сущностей 1:1 и 1: многим
Обеспечение целостности
Для чего используем Джоины?
Объединение (union)
Для чего используется Minus и Except?
Как осуществляется Группировка
Когда и как пользоваться Sum, AVG и Having

Адукар

Домашнее задание

По первой части:
1) Грабер М. Введение в SQL – читать главы 8, 9, 14 полностью.
2) на сайте SQL-EX.RU выполнить к следующему занятию задачи 6, 7, 8 (решить через JOIN), 9, 16, 23, 34, 35, 36, 38, 40, 44, 45, 48, 49, 50.

По второй части (Агрегаты)

1) читать презентации!
2) Грабер М. Введение в SQL – главы 6, 7, 10, 12, 15 полностью;
3) на сайте SQL-EX.RU выполнить к следующему занятию задачи с 1 по 27 включительно, 37, 39, 55, 56, 71, 80, 91.

http://2sql.ru/novosti/sql-having/ - тут можно найти теорию

АДУКАР