



Кафедра ЦТ
Институт информационных технологий
РТУ МИРЭА



Дисциплина «Разработка баз данных»

Практическая работа №2.

Многотабличные запросы и теоретико-множественные операции.



Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы:

Задание 1: демонстрация различных типов соединений.

1. Запрос с **INNER JOIN**: подсчитайте количество связанных записей между таблицами
(например, «сколько лекарств у каждого производителя?»)
2. Запрос с **LEFT JOIN**: проанализируйте наличие или отсутствие связей
(например, «сколько лекарств у каждого производителя, включая тех, у кого лекарств нет?»)
3. Запрос с **RIGHT JOIN** и **WHERE... IS NULL** (паттерн «анти-соединение»): найдите и подсчитайте записи без связей
(например, «сколько лекарств не имеют производителя в базе?»)
4. Запрос с **FULL JOIN**: получите **общую статистику** – сколько всего связанных записей, и сколько записей без связей.
5. Запрос с **CROSS JOIN**: сформировать декартово произведение всех записей одной таблицы со всеми записями другой, создав тем самым **все возможные комбинации строк** между ними.

(продолжение на следующем слайде)

Практическая работа №2.

Многотабличные запросы и теоретико-множественные операции.



Постановка задачи: основываясь на индивидуальной схеме данных, составьте необходимые запросы:

Задание 2: применение теоретико-множественных операторов.

На основе индивидуальной схемы данных составить и выполнить три запроса, демонстрирующих практическое применение операторов **UNION**, **INTERSECT** и **EXCEPT**.

1. **UNION:** составить единый список из данных двух разных таблиц (столбцы должны быть совместимы по типу).
2. **INTERSECT:** найти общие записи, которые удовлетворяют двум разным условиям или находятся в двух разных наборах данных.
3. **EXCEPT:** найти записи, которые присутствуют в одном наборе данных, но отсутствуют в другом.

(начало на предыдущем слайде)

1. Основы соединения таблиц (JOIN)



Изучаемый ранее процесс нормализации приводит к **разделению** данных на несколько логических сущностей (**таблиц**).

Операторы **JOIN** являются фундаментальным механизмом SQL, который позволяет «**собирать**» эти разделённые данные обратно в **единое осмысленное представление**, для дальнейшего анализа и формирования отчетов.

1. Основы соединения таблиц (JOIN)



Для извлечения данных из **нескольких таблиц** используется оператор **JOIN**.

Общий синтаксис:

SELECT

table1.column,
table2.column

FROM

table1

[**INNER** | **LEFT** | **RIGHT** | **FULL**] **JOIN**

table2

ON

table1.common_column =
table2.common_column

[**WHERE** *condition*];

Краткое описание:

SELECT – указывает столбцы из обеих таблиц, которые нужно выбрать. **Крайне рекомендуется** указывать **таблицу** (например, *table1.column*), чтобы избежать неоднозначности.

FROM – указывает первую («**левую**») таблицу для соединения.

JOIN – указывает вторую («**правую**») таблицу, которую нужно присоединить. Тип соединения (**INNER**, **LEFT** и т.д.) определяет, как именно будут объединены строки.

ON – указывает условие соединения. Это ключевая часть, которая определяет, по какому **общему полю** таблицы связаны друг с другом.

1. Основы соединения таблиц (JOIN)



INNER JOIN (можно сократить до **JOIN**)

Возвращает только те строки, для которых найдено совпадение в **обеих** таблицах.

SELECT

[нужные колонки]

FROM

Таблица 1

INNER JOIN

Таблица 2

ON

"Таблица 1".id = "Таблица 2".id

Таблица 1

X1	1
X2	2
X3	3

Таблица 2

1	Y1
4	Y2
2	Y3

INNER JOIN

X1	1	Y1
X2	2	Y3

1. Основы соединения таблиц (JOIN)



LEFT JOIN

Возвращает **все** строки из **левой** таблицы и только **совпадающие** из **правой**.

SELECT

[нужные колонки]

FROM

Таблица 1

LEFT JOIN

Таблица 2

ON

"Таблица 1".id = "Таблица 2".id

Таблица 1

X1	1
X2	2
X3	3

Таблица 2

1	Y1
4	Y2
2	Y3

LEFT JOIN

X1	1	Y1
X2	2	Y3
X3	3	NULL

1. Основы соединения таблиц (JOIN)



RIGHT JOIN

Возвращает **все** строки из **правой** таблицы и только **совпадающие** из **левой**.

SELECT

[нужные колонки]

FROM

Таблица 1

RIGHT JOIN

Таблица 2

ON

"Таблица 1".id = "Таблица 2".id

Таблица 1

X1	1
X2	2
X3	3

Таблица 2

1	Y1
4	Y2
2	Y3

RIGHT JOIN

X1	1	Y1
X2	2	Y3
NULL	4	Y2

1. Основы соединения таблиц (JOIN)



FULL JOIN

Возвращает **все** строки из **обеих** таблиц, соединяя их там, где это возможно.

SELECT

[нужные колонки]

FROM

Таблица 1

FULL JOIN

Таблица 2

ON

"Таблица 1".id = "Таблица 2".id

Таблица 1

X1	1
X2	2
X3	3

Таблица 2

1	Y1
4	Y2
2	Y3

FULL JOIN

X1	1	Y1
X2	2	Y3
X3	3	NULL
NULL	4	Y2

1. Основы соединения таблиц (JOIN)



Также существует оператор **CROSS JOIN** (декартово произведение):

Общий синтаксис:

SELECT

table1.column,
table2.column

FROM

table1

CROSS JOIN

table2;

Краткое описание:

CROSS JOIN: создаёт декартово произведение – каждая строка из первой таблицы соединяется с каждой строкой из второй.

ВАЖНО: для **CROSS JOIN** не используется предложение **ON**.

1. Основы соединения таблиц (JOIN)



CROSS JOIN

Возвращает **декартово произведение** – все возможные комбинации строк из **обеих** таблиц.

SELECT

[нужные колонки]

FROM

Таблица 1

CROSS JOIN

Таблица 2

Таблица 1

X1	1
X2	2
X3	3

Таблица 2

1	Y1
4	Y2
2	Y3

CROSS JOIN

X1	1	1	Y1
X1	1	4	Y2
X1	1	2	Y3
X2	2	1	Y1
X2	2	4	Y2
X2	2	2	Y3
X3	3	1	Y1
X3	3	4	Y2
X3	3	2	Y3

1. Основы соединения таблиц (JOIN)



Псевдонимы (**aliases**) дают временные, короткие имена таблицам или столбцам в запросе, за счёт чего повышается читаемость и уменьшается длина кода.

Псевдонимы абсолютно **необходимы** при использовании «само-соединений» (**Self-Join**), когда таблица ссылается **сама на себя**.

ВАЖНО: Для **Self-Join** не существует отдельного ключевого слова – это **паттерн** связи таблиц.

```
-- Этот код синтаксически невозможен без псевдонимов 'e' и 'm'  
SELECT  
    ...  
FROM  
    pharmacists AS e  -- Таблица в роли "сотрудников"  
LEFT JOIN  
    pharmacists AS m  -- Та же таблица в роли "менеджеров"  
ON  
    e.manager_id = m.pharmacist_id;
```

1. Основы соединения таблиц (JOIN)



SELECT

```
e.id AS id, e.имя AS имя_сотрудника,  
e.фамилия AS фамилия_сотрудника,  
e.id_руководителя, m.имя AS имя_руководителя,  
m.фамилия AS фамилия_руководителя
```

FROM сотрудники e

JOIN сотрудники m **ON** e.id_руководителя = m.id;

Таблица "Сотрудники"

ID	Имя	Фамилия	ID руководителя
1	Иван	Сорокин	NULL
2	Мария	Петровна	1
3	Алексей	Орлов	2
4	Евгений	Смирнов	1

SELF JOIN

Таблица "Сотрудники с указанием руководителя"

ID	Имя сотрудника	Фамилия сотрудника	ID руководителя	Имя руководителя	Фамилия руководителя
2	Мария	Петровна	1	Иван	Сорокин
3	Алексей	Орлов	2	Мария	Петровна
4	Евгений	Смирнов	1	Иван	Сорокин

2. Продвинутые техники и паттерны соединений



Для получения комплексных отчетов часто требуется **соединять более двух таблиц**.

Чтобы решить эту задачу, СУБД **последовательно** выполняет **соединения**:

- Соединяются **первые две** таблицы, и их результат рассматривается **как одна** временная, **виртуальная таблица**.
- Затем эта виртуальная таблица **соединяется со следующей таблицей**, формируя новую **виртуальную таблицу**. *При необходимости, этот шаг повторяется.*

Ограничений на количество соединяемых таблиц **нет**.

2. Продвинутые техники и паттерны соединений



Теоретико-множественные операторы (**UNION**, **INTERSECT**, **EXCEPT**)

Эти операторы предназначены для **объединения результирующих наборов** от двух или более **независимых SELECT-запросов** в одну итоговую таблицу.

В отличие от **JOIN**, который «склеивает» таблицы **по горизонтали** (добавляя столбцы), эти операторы «**склеивают**» результаты **по вертикали** (добавляя **строки**).

2. Продвинутые техники и паттерны соединений



Общие правила и принципы работы теоретико-множественных операторов:

- **Работа с результатами:** операторы применяются не к самим таблицам, а к **данным**, которые были возвращены каждым **отдельным** SELECT-запросом.
- **Совместимость столбцов:** это главное правило. **Все** SELECT-запросы в конструкции должны возвращать **одинаковое количество** столбцов, и **типы данных** этих столбцов должны быть **совместимы** (например, нельзя объединить число со строкой).
- **Имена столбцов:** итоговая таблица получает **имена столбцов** из **первого** SELECT-запроса.
- **Удаление дубликатов:** по умолчанию, операторы UNION, INTERSECT и EXCEPT удаляют **дубликаты** из итогового набора данных, аналогично работе SELECT DISTINCT. **Для сохранения дубликатов** при объединении используется **UNION ALL**.

2. Продвинутые техники и паттерны соединений



UNION объединяет результаты двух запросов в один набор данных, при этом **удаляя** все дублирующиеся строки.

Результат 1

Значение 1
Значение 2
Значение 3

Результат 2

Значение 1
Значение 4
Значение 2

UNION

SELECT name **FROM** medicines

UNION

SELECT name **FROM** manufacturers;

Значение 1
Значение 2
Значение 3
Значение 4

2. Продвинутые техники и паттерны соединений



UNION ALL как и **UNION**, объединяет результаты двух запросов в один набор данных, однако работает быстрее, так как **сохраняет дубликаты**.

SELECT name **FROM** medicines

UNION ALL

SELECT name **FROM** manufacturers;

Результат 1

Значение 1
Значение 2
Значение 3

Результат 2

Значение 1
Значение 4
Значение 2

UNION ALL

Значение 1
Значение 2
Значение 3
Значение 1
Значение 4
Значение 2

2. Продвинутые техники и паттерны соединений



INTERSECT (пересечение) возвращает только те строки, которые присутствуют в результатах **обоих** запросов. Порядок запросов **не имеет значения**.

Результат 1

Значение 1
Значение 2
Значение 3

Результат 2

Значение 1
Значение 4
Значение 2

INTERSECT

Значение 1
Значение 2

SELECT name **FROM** medicines

INTERSECT

SELECT name **FROM** manufacturers;

2. Продвинутые техники и паттерны соединений



EXCEPT (*разность*) возвращает строки из первого запроса, которых **нет** во втором.

Порядок запросов **критически важен**.

Результат 1

Значение 1
Значение 2
Значение 3

Результат 2

Значение 1
Значение 4
Значение 2

EXCEPT

Значение 3

SELECT name **FROM** medicines

EXCEPT

SELECT name **FROM** manufacturers;



Кафедра ЦТ
Институт информационных технологий
РТУ МИРЭА



Спасибо за внимание