Турсунов Баходурхон

**1. Как определяется первичный ключ в PostgreSQL и какие ограничения он включает?**

В PostgreSQL первичный ключ определяется с использованием специального типа ограничения, называемого "PRIMARY KEY constraint" (ограничение первичного ключа). Оно гарантирует уникальность значений в столбце или группе столбцов и обеспечивает быстрый доступ к данным.

**2. Может ли таблица иметь более одного первичного ключа? Почему или почему нет?**  
В реляционных базах данных, таких как PostgreSQL, таблица не может иметь более одного первичного ключа. Почему? Потому что первичный ключ представляет собой уникальный идентификатор строки в таблице, и каждая строка должна иметь только один уникальный идентификатор.

Если бы таблица имела более одного первичного ключа, это создало бы неоднозначность в определении уникальности каждой строки. Для каждой строки в таблице должен быть один уникальный идентификатор, который может быть использован для идентификации и доступа к этой строке. Если бы было несколько первичных ключей, это могло бы привести к неопределенности при поиске, сортировке и ссылке на данные в таблице.

Вместо этого в таблице может быть определено одно ограничение первичного ключа, которое определяет один или несколько столбцов в таблице как первичный ключ. Это обеспечивает уникальность идентификации строк и обеспечивает целостность данных в таблице.

**3. Как явно указать имя ограничения первичного ключа в PostgreSQL?**

В PostgreSQL вы можете явно указать имя ограничения первичного ключа при создании таблицы с помощью предложения **CONSTRAINT**. Вот как это делается:

CREATE TABLE Students (

id SERIAL,

name VARCHAR(50),

age INT,

CONSTRAINT pk\_students PRIMARY KEY (id)

);

В этом примере ограничение первичного ключа названо **pk\_students**. Вы можете выбрать любое уникальное имя для ограничения, которое соответствует правилам именования в PostgreSQL. Имя ограничения может быть полезно для идентификации и управления ограничениями в базе данных, особенно если вы хотите явно удалять или изменять ограничения в будущем.

Также можно определить ограничение первичного ключа отдельно от определения столбца, используя ключевое слово **CONSTRAINT** в команде **ALTER TABLE**:

ALTER TABLE Students ADD CONSTRAINT pk\_students PRIMARY KEY (id);

Этот подход позволяет добавлять или изменять ограничения первичного ключа после создания таблицы.

4. Каково назначение ограничения внешнего ключа в PostgreSQL?

Ограничение внешнего ключа в PostgreSQL используется для создания связи между двумя таблицами в базе данных. Оно определяет ссылочную целостность данных, обеспечивая связь между значением одного или нескольких столбцов в одной таблице (дочерней таблице) и значением первичного ключа или уникального ключа в другой таблице (родительской таблице).

Основное назначение ограничения внешнего ключа в PostgreSQL:

1. **Обеспечение целостности данных:** Ограничение внешнего ключа гарантирует, что значение в столбце дочерней таблицы ссылается на существующее значение в столбце родительской таблицы. Это предотвращает возможность создания "сиротских" записей в дочерней таблице, которые не имеют соответствующих записей в родительской таблице.

2. **Поддержка связей между таблицами:** Ограничение внешнего ключа позволяет создавать связи между таблицами, что облегчает выполнение операций объединения (JOIN) и обеспечивает целостность связанных данных.

3. **Автоматическая проверка целостности:** При попытке вставить, обновить или удалить запись в дочерней таблице PostgreSQL автоматически проверяет, существует ли соответствующая запись в родительской таблице. Если такая запись отсутствует, операция будет отклонена, что помогает поддерживать целостность данных.

Пример создания ограничения внешнего ключа в PostgreSQL:

CREATE TABLE Orders (

order\_id SERIAL PRIMARY KEY,

product\_id INT,

quantity INT,

CONSTRAINT fk\_product FOREIGN KEY (product\_id) REFERENCES Products (product\_id)

);

В этом примере ограничение внешнего ключа `fk\_product` связывает столбец `product\_id` в таблице `Orders` со столбцом `product\_id` в таблице `Products`.

**5. Как внешний ключ помогает поддерживать ссылочную целостность между таблицами?**

Внешний ключ помогает поддерживать ссылочную целостность между таблицами, обеспечивая связь между данными в них. Он играет ключевую роль в поддержании целостности данных путем выполнения следующих функций:

1. **Гарантирует существование связанных данных:** Внешний ключ гарантирует, что значение в столбце дочерней таблицы ссылается на существующее значение в столбце родительской таблицы. Это предотвращает возможность создания "сиротских" записей в дочерней таблице, которые не имеют соответствующих записей в родительской таблице.
2. **Запрет на удаление родительских записей:** Если в таблице существуют связанные записи в дочерних таблицах, то удаление соответствующих родительских записей нежелательно, так как это может нарушить ссылочную целостность данных. Внешний ключ может быть настроен с опцией **ON DELETE** для определения действия, которое должно быть выполнено при попытке удаления родительской записи. Например, опция **ON DELETE CASCADE** позволяет автоматически удалить все связанные записи в дочерней таблице при удалении родительской записи.
3. **Запрет на обновление родительских значений:** Если значение первичного ключа в родительской таблице изменяется, то внешний ключ может быть настроен с опцией **ON UPDATE** для определения действия, которое должно быть выполнено для связанных записей в дочерних таблицах. Например, опция **ON UPDATE CASCADE** позволяет автоматически обновить все связанные значения в дочерних таблицах при обновлении значения первичного ключа в родительской таблице.
4. **Проверка ссылочной целостности при вставке и обновлении данных:** При попытке вставить или обновить данные в столбце, содержащем внешний ключ, PostgreSQL автоматически проверяет, существует ли соответствующее значение в родительской таблице. Если такое значение отсутствует, операция будет отклонена, что помогает поддерживать ссылочную целостность данных.

Таким образом, внешний ключ является мощным инструментом для обеспечения целостности данных между таблицами, предотвращая нарушения связей между данными и гарантируя их согласованность.

**6. Объясните характеристики отношения "один к одному" в реляционной базе данных.**

Отношение "один к одному" (One-to-One) в реляционной базе данных означает, что каждая запись в одной таблице (таблица A) может быть связана с не более чем одной записью в другой таблице (таблица B), и наоборот. Это означает, что для каждой записи в таблице A существует только одна соответствующая запись в таблице B, и наоборот.

Характеристики отношения "один к одному" в реляционной базе данных включают:

1. **Уникальность связи:** Каждая запись в таблице A связана с не более чем одной записью в таблице B, и каждая запись в таблице B связана с не более чем одной записью в таблице A. Это обеспечивает уникальность связи между записями двух таблиц.
2. **Определение через первичные ключи:** Обычно отношение "один к одному" определяется через первичные ключи в каждой из таблиц. Например, в таблице A первичный ключ может быть связан с внешним ключом в таблице B и наоборот.
3. **Гибкость в моделировании:** Отношение "один к одному" может использоваться для моделирования различных типов связей в базе данных, таких как связь между сущностями и их атрибутами, разделение сущностей для нормализации базы данных и другие.
4. **Эффективное использование памяти:** Так как каждая запись связана только с одной записью в другой таблице, отношение "один к одному" может быть эффективным с точки зрения использования памяти и ресурсов базы данных.

Примером отношения "один к одному" может служить связь между таблицами "Пользователи" и "Профили", где каждый пользователь имеет только один профиль, и каждый профиль принадлежит только одному пользователю.

**7. Что отличает отношения "один ко многим" от отношений "один к одному"? Приведите пример.**

Отношение "один ко многим" (One-to-Many) и отношение "один к одному" (One-to-One) отличаются по тому, как связаны записи между таблицами.

1. **Отношение "один ко многим" (One-to-Many):**
   * В отношении "один ко многим" каждая запись в одной таблице может быть связана с одной или несколькими записями в другой таблице.
   * Это означает, что для каждой записи в таблице A может существовать несколько соответствующих записей в таблице B, но для каждой записи в таблице B существует только одна соответствующая запись в таблице A.
   * Пример: Одна книга может иметь несколько авторов, но каждый автор может быть связан только с одной книгой.
2. **Отношение "один к одному" (One-to-One):**
   * В отношении "один к одному" каждая запись в одной таблице может быть связана с не более чем одной записью в другой таблице, и наоборот.
   * Это означает, что для каждой записи в таблице A существует только одна соответствующая запись в таблице B, и наоборот.
   * Пример: Каждый пользователь может иметь только один профиль, и каждый профиль принадлежит только одному пользователю.

**Пример:** Предположим, у нас есть две таблицы: "Пользователи" и "Заказы".

* В случае отношения "один к одному" каждый пользователь может иметь только один заказ, а каждый заказ принадлежит только одному пользователю. Это может быть полезно, например, когда каждый пользователь может иметь только один текущий заказ.
* В случае отношения "один ко многим" каждый пользователь может размещать несколько заказов, но каждый заказ принадлежит только одному пользователю. Это может быть полезно, например, когда каждый пользователь может делать неограниченное количество заказов.

8. Чем отношения "многие-ко-многим" отличаются от отношений "один-к-одному" и "один-ко-многим"?

Отношения "многие-ко-многим" (Many-to-Many) отличаются от отношений "один-к-одному" (One-to-One) и "один-ко-многим" (One-to-Many) особенностями связи между записями в базе данных.

Вот как отличаются эти типы отношений:

1. **Многие-ко-многим (Many-to-Many):**
   * В отношении "многие-ко-многим" каждая запись в одной таблице может быть связана с несколькими записями в другой таблице, и наоборот.
   * Это означает, что каждая запись в таблице A может иметь несколько соответствующих записей в таблице B, и каждая запись в таблице B может иметь несколько соответствующих записей в таблице A.
   * Для моделирования отношения "многие-ко-многим" обычно используется промежуточная таблица, которая устанавливает связи между записями в обеих таблицах.
   * Пример: В базе данных музыкального магазина каждый альбом может иметь несколько исполнителей, и каждый исполнитель может участвовать в нескольких альбомах.
2. **Один-к-одному (One-to-One):**
   * В отношении "один-к-одному" каждая запись в одной таблице может быть связана только с одной записью в другой таблице, и наоборот.
   * Это означает, что каждая запись в таблице A связана с не более чем одной записью в таблице B, и каждая запись в таблице B связана с не более чем одной записью в таблице A.
   * Пример: Каждый пользователь может иметь только один профиль, и каждый профиль принадлежит только одному пользователю.
3. **Один-ко-многим (One-to-Many):**
   * В отношении "один-ко-многим" каждая запись в одной таблице может быть связана с несколькими записями в другой таблице, но каждая запись во второй таблице может быть связана только с одной записью в первой таблице.
   * Это означает, что для каждой записи в таблице A может существовать несколько соответствующих записей в таблице B, но для каждой записи в таблице B существует только одна соответствующая запись в таблице A.
   * Пример: Каждый автор может написать несколько книг, но каждая книга принадлежит только одному автору.

Таким образом, основное различие между этими типами отношений заключается в количестве записей, которые могут быть связаны между собой в каждой таблице.

**9. Объясните процесс INNER JOIN в реляционной базе данных. Как происходит объединение данных из двух таблиц?**

INNER JOIN в реляционной базе данных используется для объединения данных из двух таблиц на основе совпадения значений в указанных столбцах. Он возвращает только те строки, для которых есть соответствующие значения в обеих таблицах, удовлетворяющие условию объединения.

Процесс INNER JOIN включает следующие шаги:

1. **Определение таблиц для объединения:** Сначала определяются две таблицы, которые будут объединены. Обычно это делается с помощью ключевого слова **FROM**, указывая имена таблиц.
2. **Условие объединения:** Затем определяется условие объединения, которое определяет, какие строки из каждой таблицы должны быть объединены. Условие объединения указывается с помощью ключевого слова **ON** и может быть базовым условием равенства между столбцами в обеих таблицах.
3. **Выбор соответствующих строк:** INNER JOIN сравнивает значения в указанных столбцах каждой строки из первой таблицы с значениями в указанных столбцах каждой строки из второй таблицы. Если значения совпадают в обеих таблицах, строки объединяются и добавляются в результирующий набор данных.
4. **Формирование результирующего набора данных:** После выполнения сравнения и выборки соответствующих строк формируется результирующий набор данных, который содержит столбцы из обеих таблиц и строки, удовлетворяющие условию объединения.

Пример SQL запроса с INNER JOIN:

SELECT Orders.order\_id, Customers.customer\_name FROM Orders

INNER JOIN Customers ON Orders.customer\_id = Customers.customer\_id;

В этом примере происходит INNER JOIN между таблицами "Orders" и "Customers" на основе равенства значений в столбцах "customer\_id". Результирующий набор данных будет содержать столбцы **order\_id** из таблицы "Orders" и **customer\_name** из таблицы "Customers" для тех строк, где значения **customer\_id** совпадают в обеих таблицах.

**10. Приведите пример использования INNER JOIN с таблицами A и B? Включите предложения SELECT, FROM и JOIN.**

SELECT A.column1, A.column2, B.column3 FROM TableA AS A

INNER JOIN TableB AS B ON A.common\_column = B.common\_column;

В этом примере:

* **SELECT A.column1, A.column2, B.column3** определяет, какие столбцы будут выбраны в результирующем наборе данных. В данном случае выбираются столбцы **column1** и **column2** из таблицы A, а также **column3** из таблицы B.
* **FROM TableA AS A** и **FROM TableB AS B** определяют таблицы, которые участвуют в запросе. Здесь **TableA** и **TableB** - это названия таблиц, а **AS A** и **AS B** - псевдонимы, которые используются для упрощения ссылок на эти таблицы в запросе.
* **INNER JOIN** указывает, что мы выполняем внутреннее объединение (INNER JOIN). Он объединяет строки из таблицы A с соответствующими строками из таблицы B на основе условия, указанного в предложении **ON**.
* **ON A.common\_column = B.common\_column** определяет условие объединения. Здесь **common\_column** - это общий столбец, значения которого сравниваются для определения соответствия строк из обеих таблиц.

Этот запрос вернет результирующий набор данных, содержащий столбцы **column1** и **column2** из таблицы A, а также столбец **column3** из таблицы B для тех строк, где значения общего столбца **common\_column** совпадают в обеих таблицах.

**11. Опишите результат FULL OUTER JOIN, когда есть совпадение между строками в объединенных таблицах и когда его нет.**

FULL OUTER JOIN возвращает все строки из обеих таблиц, объединяя строки из каждой таблицы на основе условия объединения. Результат FULL OUTER JOIN включает в себя:

**1. Совпадение строк:**

- Если в таблице A и B есть строки, для которых условие объединения выполняется (то есть значения в соответствующих столбцах равны), эти строки будут объединены в одну строку в результирующем наборе данных. Такие строки будут содержать значения из обеих таблиц.

**2. Несовпадение строк:**

- Если в одной из таблиц есть строки, для которых нет соответствующих строк в другой таблице, эти строки также будут включены в результирующий набор данных. В этом случае для отсутствующих строк в другой таблице будут добавлены значения NULL.

Таким образом, результат FULL OUTER JOIN содержит все строки из обеих таблиц, включая строки, которые имеют совпадение и те, которые не имеют. Если для строки в одной таблице нет соответствующей строки в другой таблице, соответствующие столбцы будут содержать значения NULL.

Пример:

Пусть у нас есть две таблицы A и B:

```

Таблица A:

| id | name |

|----|--------|

| 1 | Alice |

| 2 | Bob |

| 3 | Charlie|

Таблица B:

| id | age |

|----|-----|

| 1 | 25 |

| 3 | 30 |

| 4 | 35 |

```

Результат FULL OUTER JOIN между этими таблицами будет:

```

| id | name | age |

|----|--------|-----|

| 1 | Alice | 25 |

| 2 | Bob | NULL|

| 3 | Charlie| 30 |

| 4 | NULL | 35 |

```

Здесь строки с id 1 и 3 имеют совпадение между таблицами, а строки с id 2 и 4 не имеют.

**12. На приведенной диаграмме Венна для FULL OUTER JOIN, как различные области диаграммы связаны с результатом запроса?**

На диаграмме Венна для FULL OUTER JOIN различные области диаграммы представляют различные наборы данных, которые включаются в результат запроса. Рассмотрим каждую область:

**1. Область, представляющая только таблицу A (левая область):**

- В этой области содержатся строки из таблицы A, для которых нет соответствующих строк в таблице B. Это означает, что все значения из столбцов таблицы A будут включены в результат запроса, а значения из столбцов таблицы B будут NULL.

**2. Область, представляющая только таблицу B (правая область):**

- В этой области содержатся строки из таблицы B, для которых нет соответствующих строк в таблице A. В результате запроса будут включены все значения из столбцов таблицы B, а значения из столбцов таблицы A будут NULL.

**3. Область, представляющая пересечение обеих таблиц (центральная область):**

- В этой области содержатся строки из таблицы A и B, для которых существует соответствие по условию объединения. Эти строки будут включены в результат запроса, и они будут содержать значения из обоих таблиц.

**13. Объясните процесс LEFT JOIN, в частности, как он обрабатывает совпадающие и не совпадающие значения между таблицами A и B.**

LEFT JOIN в реляционной базе данных используется для объединения строк из таблицы A со всеми соответствующими строками из таблицы B на основе условия объединения. В случае отсутствия соответствующих строк в таблице B, значения в результирующем наборе данных для столбцов из таблицы B будут NULL.

Процесс LEFT JOIN включает следующие этапы:

1. **Выбор всех строк из таблицы A:**
   * Сначала выбираются все строки из таблицы A.
2. **Объединение строк из таблицы B:**
   * Для каждой строки из таблицы A выполняется поиск соответствующих строк в таблице B на основе условия объединения.
   * Если найдены соответствующие строки в таблице B, они объединяются с текущей строкой из таблицы A.
   * Если для строки из таблицы A нет соответствующих строк в таблице B, она также остается в результирующем наборе данных, но значения столбцов из таблицы B будут NULL.

Таким образом, LEFT JOIN обрабатывает совпадающие и несовпадающие значения между таблицами A и B следующим образом:

* **Совпадающие значения:**
  + Строки из таблицы A, для которых найдены соответствующие строки в таблице B, объединяются с этими строками из таблицы B.
* **Несовпадающие значения:**
  + Строки из таблицы A, для которых не найдены соответствующие строки в таблице B, также остаются в результирующем наборе данных, но значения столбцов из таблицы B будут NULL.

Это позволяет LEFT JOIN сохранять все строки из таблицы A в результирующем наборе данных, включая те, для которых нет соответствующих записей в таблице B.

**14. Используйте диаграмму Венна для LEFT JOIN, чтобы проиллюстрировать сценарии, когда создается новая строка и когда заполняются значения NULL.**

Для использования диаграммы Венна для LEFT JOIN, давайте предположим, что у нас есть две таблицы: A и B. Рассмотрим сценарии, когда создается новая строка и когда заполняются значения NULL в результате LEFT JOIN.

Предположим, что таблица A содержит следующие данные:

```

Таблица A:

| id | name |

|----|--------|

| 1 | Alice |

| 2 | Bob |

| 3 | Charlie|

```

И таблица B содержит следующие данные:

```

Таблица B:

| id | age |

|----|-----|

| 1 | 25 |

| 3 | 30 |

| 4 | 35 |

```

Теперь давайте построим диаграмму Венна для LEFT JOIN:

```

Таблица A LEFT JOIN Таблица B

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| | | |

| | Совпадение строк | |

| id | id age | age |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|

| 1 | 1 25 | 25 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|

| 2 | NULL NULL | NULL |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|

| 3 | 3 30 | 30 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|

| | | |

| Новая строка | | |

| | | |

| 4 | NULL NULL | 35 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|

```

На этой диаграмме:

**1. Совпадение строк:**

- Строки с id 1 и 3 имеют совпадение между таблицами A и B. Эти строки объединяются по LEFT JOIN, и значения age из таблицы B добавляются в результирующий набор данных.

**2. Новая строка:**

- Строка с id 2 из таблицы A не имеет соответствия в таблице B. Однако она все равно остается в результирующем наборе данных, и значения age заполняются NULL.

Таким образом, диаграмма Венна для LEFT JOIN показывает, как создаются новые строки и как заполняются значения NULL в результате объединения таблицы A с таблицей B.

**15. Опишите процесс RIGHT JOIN, подчеркнув, как он обрабатывает совпадающие и не совпадающие значения.**

RIGHT JOIN в реляционной базе данных используется для объединения строк из таблицы B со всеми соответствующими строками из таблицы A на основе условия объединения. В случае отсутствия соответствующих строк в таблице A, значения в результирующем наборе данных для столбцов из таблицы A будут NULL.

Процесс RIGHT JOIN включает следующие этапы:

1. **Выбор всех строк из таблицы B:**
   * Сначала выбираются все строки из таблицы B.
2. **Объединение строк из таблицы A:**
   * Для каждой строки из таблицы B выполняется поиск соответствующих строк в таблице A на основе условия объединения.
   * Если найдены соответствующие строки в таблице A, они объединяются с текущей строкой из таблицы B.
   * Если для строки из таблицы B нет соответствующих строк в таблице A, она также остается в результирующем наборе данных, но значения столбцов из таблицы A будут NULL.

Таким образом, RIGHT JOIN обрабатывает совпадающие и несовпадающие значения между таблицами A и B следующим образом:

* **Совпадающие значения:**
  + Строки из таблицы B, для которых найдены соответствующие строки в таблице A, объединяются с этими строками из таблицы A.
* **Несовпадающие значения:**
  + Строки из таблицы B, для которых не найдены соответствующие строки в таблице A, также остаются в результирующем наборе данных, но значения столбцов из таблицы A будут NULL.

Это позволяет RIGHT JOIN сохранять все строки из таблицы B в результирующем наборе данных, включая те, для которых нет соответствующих записей в таблице A.

**16. Используя диаграмму Венна для RIGHT JOIN, объясните, в каких ситуациях создаются строки и как обрабатываются значения NULL.**

Давайте воспользуемся диаграммой Венна для RIGHT JOIN, чтобы проиллюстрировать ситуации, когда создаются строки и как обрабатываются значения NULL.

Предположим, что у нас есть две таблицы: A и B. Рассмотрим диаграмму Венна для RIGHT JOIN:

Таблица A RIGHT JOIN Таблица B

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| | | |

| id | Совпадение строк | id |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|

| 1 | 1 age1 | 1 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|

| NULL | 2 NULL | 2 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|

| 3 | NULL NULL | 3 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|

| | | |

| Новая строка | | |

| | | |

| NULL | 4 age4 | 4 |

|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_|

```

На этой диаграмме:

**1. Совпадение строк:**

- Строки с id 1 и 3 из таблицы B имеют совпадение между таблицами A и B. Эти строки объединяются по RIGHT JOIN, и значения age из таблицы A добавляются в результирующий набор данных.

**2. Новая строка:**

- Строка с id 2 из таблицы B не имеет соответствия в таблице A. Однако она все равно остается в результирующем наборе данных, и значения id и age из таблицы A заполняются NULL.

**3. Строки из таблицы B без совпадений:**

- Строка с id 4 из таблицы B не имеет соответствующих строк в таблице A. Эта строка также остается в результирующем наборе данных, и значения id из таблицы A заполняются NULL.

Таким образом, в результате RIGHT JOIN создаются строки, когда нет совпадающих значений, и значения NULL заполняются для столбцов из таблицы A, которые не имеют соответствия в таблице B.

Practice:

1. Создайте две таблицы, сотрудники и отделы, в каждой из которых есть соответствующие столбцы (например, employee\_id, department\_id, name и т. д.).

Заполните таблицы образцами данных.

-- Создание таблицы Employees

CREATE TABLE Employees (

employee\_id INT PRIMARY KEY,

employee\_name VARCHAR(50),

department\_id INT

);

-- Вставка образцов данных в таблицу Employees

INSERT INTO Employees (employee\_id, employee\_name, department\_id)

VALUES

(1, 'John Doe', 1),

(2, 'Jane Smith', 2),

(3, 'Michael Johnson', 1),

(4, 'Emily Brown', NULL);

-- Создание таблицы Departments

CREATE TABLE Departments (

department\_id INT PRIMARY KEY,

department\_name VARCHAR(50)

);

-- Вставка образцов данных в таблицу Departments

INSERT INTO Departments (department\_id, department\_name)

VALUES

(1, 'HR'),

(2, 'Finance'),

(3, 'IT');

2. Напишите SQL-запрос с использованием INNER JOIN, чтобы получить список сотрудников и соответствующие им названия отделов.

Включите такие столбцы, как employee\_id, employee\_name, department\_id и department\_name.

Убедитесь, что в набор результатов включены только совпадающие записи.

SELECT

Employees.employee\_id,

Employees.employee\_name,

Employees.department\_id,

Departments.department\_name

FROM

Employees

INNER JOIN

Departments ON Employees.department\_id = Departments.department\_id;

3. Установите ПОЛНУЮ ВНЕШНЮЮ СВЯЗЬ между таблицами Employees и Departments.

Выведите список всех сотрудников и информацию об их отделах, независимо от того, есть ли совпадение.

Включите столбцы для обеих таблиц и соответствующим образом обработайте значения NULL.

SELECT

Employees.employee\_id,

Employees.employee\_name,

Employees.department\_id,

Departments.department\_name

FROM

Employees

FULL OUTER JOIN

Departments ON Employees.department\_id = Departments.department\_id;

4. Используйте LEFT JOIN для поиска сотрудников, у которых может совпадать или не совпадать отдел.

Получите такие столбцы, как employee\_id, employee\_name и department\_name.

Решите проблемы, когда для сотрудника может не быть соответствующего отдела.

SELECT

Employees.employee\_id,

Employees.employee\_name,

Departments.department\_name

FROM

Employees

LEFT JOIN

Departments ON Employees.department\_id = Departments.department\_id;

5. Используйте RIGHT JOIN для получения данных об отделах и связанных с ними сотрудниках.

Включите такие столбцы, как department\_id, department\_name, и информацию, связанную с сотрудниками.

Разберитесь с ситуацией, когда может существовать отдел без назначенных сотрудников.

SELECT

Employees.employee\_id,

Employees.employee\_name,

Departments.department\_id,

Departments.department\_name

FROM

Departments

RIGHT JOIN

Employees ON Employees.department\_id = Departments.department\_id;

Доп задание:

1. Создать обязательную зависимость таблицы Risk с RiskProfile