

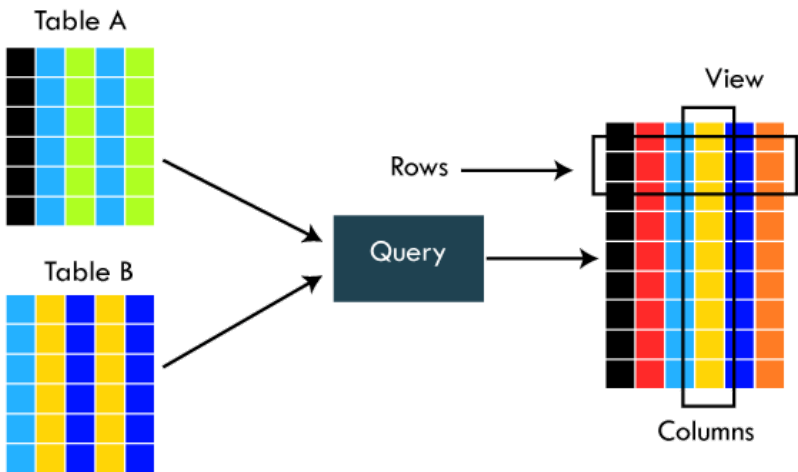
Представления (view)

Представления, известные как VIEW, представляют собой мощный инструмент для управления и представления данных в удобной и безопасной форме.

Чтобы понять, что такое view и зачем они нужны, представьте себе ситуацию, где у вас есть сложная база данных с множеством таблиц, содержащих большое количество информации. Представления позволяют вам создать своеобразное "окно" в эту базу данных, через которое можно увидеть только те данные, которые вам нужны, и именно в том виде, в каком они вам нужны.

Предположим, у вас есть большая таблица с данными о сотрудниках компании, включая их имена, отделы, зарплаты и другие личные данные. Теперь представьте, что вам нужно регулярно создавать отчеты только с именами и отделами сотрудников, без включения чувствительных данных, таких как зарплата или личная информация. Вместо того чтобы каждый раз писать сложный запрос к базе данных, вы можете создать view, который будет автоматически выбирать только нужные столбцы.

Можно собирать данные даже из разных таблиц:



Какие view бывают?

1. Обычные

Обычные представления, или стандартные view, это виртуальные таблицы, формируемые на основе результата SQL-запроса. Они не хранят физически данные, а предоставляют динамический доступ к данным, вычисляемым при каждом обращении к view. Это означает, что данные в обычном view всегда актуальны и отражают текущее состояние основной таблицы или таблиц, из которых они формируются.

Рассмотрим таблицу `employees` , которая содержит подробные данные о сотрудниках, включая их ID, имя, отдел, зарплату и прочее. Допустим, вам нужно часто получать список сотрудников и отделов, в которых они работают, без дополнительной чувствительной информации, такой как зарплата. Вы можете создать обычное представление:

```
CREATE VIEW employee_department AS
SELECT employee_id, name, department FROM employees;
```

Теперь, когда потребуется получить список сотрудников и их отделов, вы просто обращаетесь к view, а не таблице:

```
SELECT * FROM employee_department;
```

Это представление будет динамически формировать данные на основе текущего состояния таблицы `employees` . То есть каждый раз при запросе к view, будет выполняться запрос к таблице (SELECT employee_id, name, department FROM employees;).

2. Материализованные View (Materialized Views)

Материализованные view, в отличие от обычных, физически хранят данные. Они особенно полезны, когда необходимо работать с большими объемами данных или выполнить сложные расчеты, которые занимают значительное время. Материализованные view обновляются не автоматически, а по команде или в соответствии с заданным расписанием.

Предположим, у вас есть таблица `sales` , содержащая информацию о продажах продуктов. Вы хотите быстро получать сводную информацию о продажах по каждому продукту. Создаем материализованное представление:

```
CREATE MATERIALIZED VIEW sales_summary AS
SELECT product_id, SUM(amount) AS total_sales
FROM sales
GROUP BY product_id;
```

Теперь, для получения сводки по продажам, достаточно обратиться к этому view:

```
SELECT * FROM sales_summary;
```

Это представление не будет автоматически обновляться при каждом изменении данных в `sales` , что делает его особенно полезным для аналитических запросов, где не требуется мгновенное отражение изменений в данных. А если нужно обновить данные для `view` - просто выполните отдельный запрос с командой `UPDATE` для обновления `view`.

Представления (view) в базах данных могут помочь в разных случаях:

- 1. Упрощение сложных запросов: View позволяют скрыть сложность запросов, содержащих множественные соединения, подзапросы и сложные условия. Это делает данные более доступными для пользователей, которые не знакомы с сложными SQL-запросами (например для первой линии тех.поддержки).
- 2. Обеспечение безопасности и скрытия важных данных: Представления могут ограничить доступ к определенным столбцам или строкам в таблице, тем самым защищая чувствительные данные от несанкционированного доступа.
- 3. Удобный формат данных: View позволяют представлять данные в формате, который удобнее для конкретных аналитических или отчетных задач, не меняя при этом структуру основных таблиц.
- 4. Управление версиями: View могут быть использованы для поддержания совместимости приложений при изменении структуры базы данных, позволяя старым приложениям работать с новыми данными.
- 5. Упрощение повторного использования кода: Если один и тот же сложный большой запрос используется во многих местах, его можно преобразовать в `view`, чтобы избежать дублирования кода.
- 6. Оптимизация производительности: В некоторых случаях, особенно в системах с материализованными представлениями, `view` могут ускорить выполнение запросов за счет кэширования результатов.
- 7. Скрытие изменений: Если структура основных таблиц меняется, необходимо изменить только определение `view`, а не все запросы, которые используют эти таблицы.

Будьте внимательны при работе с `view`, они не нужны везде, учитывайте разные аспекты вашей системы:

1. Производительность:

- Обычные view не хранят данные физически и выполняют запрос каждый раз при обращении к ним. Это может привести к дополнительной нагрузке на сервер, особенно если основной запрос сложный и работает с большими объемами данных.
- Материализованные view хранят данные физически, что улучшает скорость чтения данных, но требует дополнительного пространства на диске и времени на обновление данных.
- Убедитесь, что запрос, лежащий в основе view, оптимизирован. Это особенно важно для обычных view, поскольку запрос выполняется при каждом обращении к представлению.

2. Сложность:

- Используйте view там, где это действительно упрощает запросы или улучшает безопасность, но избегайте их чрезмерного использования, чтобы не усложнять архитектуру базы данных.
- Избегайте использования view для операций обновления, особенно если view основан на сложных запросах.