ЧТЕНИЕ ДАННЫХ ИЗ НЕСКОЛЬКИХ ТАБЛИЦ: ОБЪЕДИНЕНИЯ (JOIN) И ВЛОЖЕННЫЕ ЗАПРОСЫ

Объединения

В реляционных базах данных информация распределена по таблицам, а большая часть работы в таких базах подразумевает поиск нужных сведений из разных таблиц одновременно.

Объединение (JOIN) – это оператор SQL, который соединяет данные из разных таблиц базы банных с целью получения нужного подмножества данных.

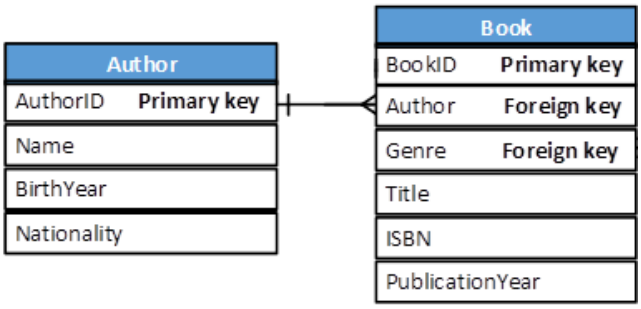
JOIN позволяет связать данные, создавая единое представление информации, распределённой по разным таблицам, на основе общих атрибутов. Это позволяет проводить более сложный анализ данных и извлекать нужную информацию из нескольких источников одновременно

Оператор JOIN один из наиболее важных в SQL. Он сочетается с командой SELECT, которая выбирает данные из таблицы, а также с операторами WHERE, ORDER BY и GROUP BY, используемыми в рамках SELECT для фильтрации, сортировки и группировки данных соответственно.

Сразу рассмотрим пример.

```
# Объединение имени автора и названия книги (таблицы Author, Book)
cursor.execute('''
    SELECT Author.Name , Book.Title
    FROM Book
    JOIN Author ON Author.AuthorID = Book.Author
''')
```

Этот случай характерен для связи «один-ко многим» где у одного авторам – много книг, но одна книга принадлежит одном автору.



```
SELECT Author.Name , Book.Title
```

это команда SQL, которая используется для извлечения данных из таблиц. В данном случае выбираются два столбца:

- Author.Name – это столбец `Name` из таблицы `Author`, который содержит имена авторов.
- Book.Title – это столбец `Title` из таблицы `Book`, который содержит названия книг.

В процессе выполнения запроса происходит полное объединение всех данных из обеих таблиц. Но в результате запроса выводятся только два выбранных столбца – Author.Name и Book.Title.

```
FROM Author
```

FROM — это ключевое слово, которое указывает, из какой таблицы будут извлекаться данные. В данном случае, данные будут извлекаться из таблицы `Author` (авторы).

```
JOIN Book ON Author.AuthorID = Book.Author
```

— это ключевое слово, которое указывает на необходимость объединения данных JOIN из двух таблиц. По умолчанию, **JOIN** представляет собой **INNER JOIN**, что означает, что он возвращает только те строки, для которых найдены совпадения в обеих таблицах.

Book – это вторая таблица, с которой выполняется объединение.

ON Author.AuthorID = Book.Author – это условие соединения. Оно указывает, как таблицы будут соединяться. Ключевое слово ON буквально означает «по условию» или «на основании условия»

Для стандартного оператора INNER JOIN порядок таблиц не имеет значения для самой логики соединения. Результат будет одинаковым, независимо от того, какая таблица указана слева, а какая справа. Главное условие, по которому происходит соединение, остаётся прежним (по полям, которые указаны после ON).

Изначально JOIN – бинарный оператор, то есть он работает с двумя переданными ему объектами – с двумя таблицами. На практике современные реализации могут воспринимать и больше двух таблиц, просто операция в таком случае выполняется несколько раз. Команде JOIN передаются таблицы, которые нужно объединить, и критерий для объединения – логическое выражение, которое называется *ключом*. Ключ может быть определен на основе значений одного или нескольких столбцов в каждой из таблиц. JOIN проверяет соответствие этих значений и формирует результат, включающий комбинацию данных из обеих таблиц, где соответствующие значения ключа совпадают.

Пример объединения данных из трех таблиц: Book, Author и Genre. :

```
cursor.execute('''
        SELECT Book.Title, Author.Name, Genre.GenreName
        FROM Book
        JOIN Author ON Book.Author = Author.AuthorID
        JOIN Genre ON Book.Genre = Genre.GenreID
    ''')
```

Каждое объединение таблиц через JOIN происходит на основе логических выражений, как показано в ON Book.Author = Author.AuthorID и ON Book.Genre = Genre.GenreID. Эти логические выражения сравнивают значения в соответствующих полях и определяют, какие строки из разных таблиц должны быть объединены.

В результате выполнения запроса будут выбраны следующие данные:

- Название книги (Book.Title),
- Имя автора (Author.Name),
- Название жанра книги (Genre.GenreName),

Порядок указания таблиц не меняет результата, потому что JOIN выполняется по внешнему ключу и первичному ключу, а не зависит от того, в каком порядке они указаны в запросе.

Если поменять порядок таблиц, результат будет тот же:

```
cursor.execute('''
SELECT Book.Title, Author.Name, Genre.GenreName, Book.PublicationYear
FROM Author
JOIN Book ON Book.Author = Author.AuthorID
JOIN Genre ON Book.Genre = Genre.GenreID
...)
```

Если таблицу Genre поставить первой в запросе, то это тоже будет работать корректно, но с учетом того, как сформулированы условия соединений.

```
cursor.execute('''
SELECT Book.Title, Author.Name, Genre.GenreName, Book.PublicationYear
FROM Genre
JOIN Book ON Book.Genre = Genre.GenreID
JOIN Author ON Book.Author = Author.AuthorID
...)
```

Реализации оператора JOIN

Оператор JOIN имеет разные режимы работы или разные реализации. Стандарт языка SQL определяет два типа объединений – внутреннее (INNER JOIN) и внешнее (OUTER JOIN). Причем внешнее объединение представляет собой семейство, состоящее из левого (LEFT OUTER JOIN или просто LEFT JOIN), правого (RIGHT OUTER JOIN или просто RIGHT JOIN), а также полного (FULL OUTER JOIN или просто FULL JOIN). Для левого и правого объединений существуют еще и условные варианты – «LEFT JOIN (if NULL)» и «RIGHT JOIN (if NULL)». Они не являются официальными частями стандарта SQL, потому что могут быть реализованы через другие конструкции языка SQL. Однако их применение оправдано в случае, когда нет совпадений в столбцах, указанных в условии JOIN.

Особняком стоят еще две команды SELF JOIN и CROSS JOIN. Строго говоря, они не являются типами JOIN. SELF JOIN относится к ситуации, когда таблица присоединяется сама к себе, обычно для сравнения значений в одной таблице. Это не совсем тип JOIN, скорее концепция использования JOIN на одной и той же таблице. CROSS JOIN – это операция, которая производит *декартово произведение двух таблиц*, возвращая результат, который является комбинацией каждой строки из первой таблицы с каждой строкой из второй таблицы.

Особенности работы каждого типа легче всего объяснить с помощью диаграмм Венна, которые также называют кругами Эйлера (рисунок 1). В контексте баз данных каждый круг обозначает отдельную таблицу, а пересечение кругов соответствует результатам операций JOIN, где строки из разных таблиц удовлетворяют определенным условиям объединения.

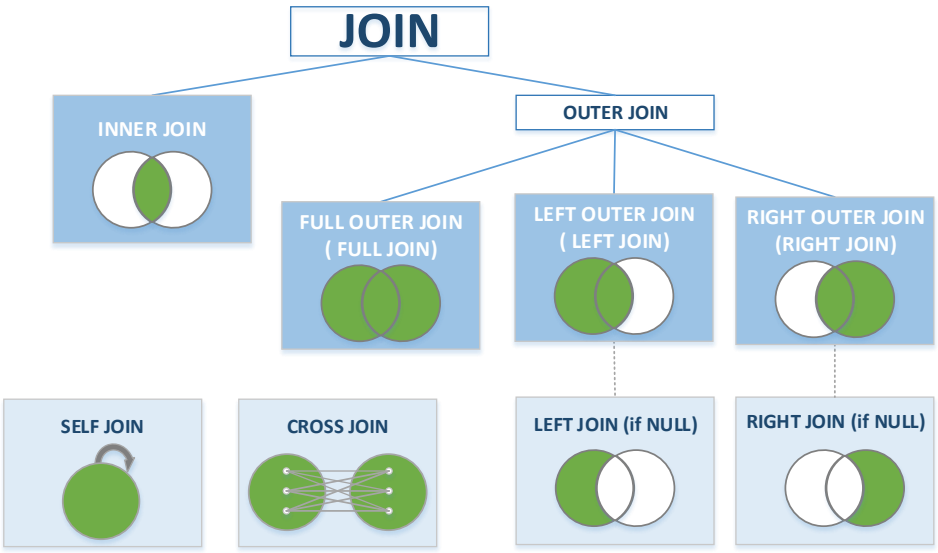


Рисунок 1 – Варианты команды JOIN

INNER JOIN – внутреннее объединение. Это самый простой и часто используемый вариант команды. Если режим работы команды явно не указан, то компилятор языка SQL автоматически воспримет его как внутренний JOIN. При использовании INNER JOIN передаются две таблицы, и оператор возвращает их внутреннее пересечение по заданному критерию объединения. Это означает, что результирующий набор данных будет содержать только записи, которые имеют соответствующие значения в обеих таблицах, основываясь на заданных условиях объединения. Такое объединение имеет сходство с логической операций «И». Например, в базе данных есть две таблицы: «Кафе» и «Рестораны». Обе таблицы содержат атрибут «Тип кухни». Если выполнить INNER JOIN между этими двумя таблицами по атрибуту «Тип кухни», то результирующий набор данных будет содержать только сходные объекты – кафе и рестораны, предлагающие одинаковую кухню. Пример SQL-запроса:

```
SELECT *
FROM Cafe
JOIN Restaurant ON Cafe.Cuisine_type = Restaurant.Cuisine_type
```

Table "Café"

ID	CafeName	Cuisine_type
1	Кафе А	Итальянская
2	Кафе Б	Французская
3	Кафе В	Японская
4	Кафе Г	Итальянская
5	Кафе Д	Русская

Table "Restaurant"

D	RestaurantName	Cuisine_type
1	Ресторан 1	Французская
2	Ресторан 2	Итальянская
3	Ресторан 3	Китайская
4	Ресторан 4	Русская
5	Ресторан 5	Китайская

Result Table

ID	CafeName	Cuisine_type	D	RestaurantName	Cuisine_type
1	Кафе А	Итальянская	2	Ресторан 2	Итальянская
4	Кафе Г	Итальянская	2	Ресторан 2	Итальянская
2	Кафе Б	Французская	1	Ресторан 1	Французская
3	Кафе Д	Русская	4	Ресторан 4	Русская

Здесь объединение производится уже не по равенству ключевых полей, а по равенству других столбцов таблицы (других атрибутов).

OUTER JOIN – внешнее соединение. Если внутреннее объединение имеет сходство с бинарным «И», то внешнее – это уже несколько вариаций логического «ИЛИ». Операторы OUTER JOIN возвращают не только строгое пересечение между двумя таблицами, но и отдельные элементы, которые принадлежат только одному из множеств.

LEFT JOIN возвращает результат, который включает в себя все строки из левой таблицы, независимо от того, есть ли соответствующие строки в правой таблице. Если для строки из левой таблицы нет соответствующей строки в правой таблице, то в результирующем наборе для соответствующих столбцов из правой таблицы будут возвращены значения NULL. Здесь левая таблица – та, которая в запросе указана первой, а правая – второй. Например, в базе данных есть две таблицы: «Клиент» и «Заказ». Клиенты могут делать заказы, и в базе данных есть информация о клиентах и их заказах. Пример SQL-запроса:

```
SELECT Client.ClientID, Client.Name, Order.OrderID, Order.Product
FROM Client
LEFT JOIN Order ON Client.ClientID = Order.ClientID
```

Table "Client"

ClientID	Name
1	Анна
2	Иван
3	Мария

Table "Order"

OrderID	ClientID	Product
101	1	Кофе
102	1	Пирожное
103	2	Чай

Result Table

ClientID	Name	OrderID	Product
1	Анна	101	Кофе
4	Анна	102	Пирожное
2	Иван	103	Чай
3	Мария	(NULL)	(NULL)

В этом запросе используется LEFT JOIN, чтобы получить **всех клиентов** вместе с их заказами. Для этого производится соединение таблицы «Клиенты» с таблицей «Заказы» по полю «ClientID». LEFT JOIN гарантирует, что все клиенты из таблицы «Клиент» будут включены в результирующий набор, даже если у них нет заказов. Если у клиента нет заказов, то для полей из таблицы «Заказы» будут возвращены значения NULL.

RIGHT JOIN вместо левой таблицы включает все строки из правой таблицы, независимо от того, есть ли соответствующие строки в левой таблице.

Если для строки из правой таблицы нет соответствующей строки в левой таблице, то в результирующем наборе для соответствующих столбцов из левой таблицы будут возвращены значения NULL. Например, в базе данных имеется таблица «Абонент», содержащая информацию о различных абонентах мобильной связи, и таблица «Тарифный план», содержащая информацию о доступных тарифных планах. Возможна ситуация, когда есть тарифные планы, которые пока еще не используются ни одним абонентом. В этом случае можно использовать RIGHT JOIN, чтобы получить список всех тарифных планов, включая те, к которым не подключены абоненты:

```
SELECT *
FROM Tariff
RIGHT JOIN Abonent ON Tariff.TariffID = Abonent.TariffID
```

Table "Abonent"

AbonentID	AbonentName	TariffID
1	Анна	2
2	Иван	3
3	Мария	1

Table "Tariff"

TariffID	TariffName	Cost	IncludedData
1	План А	300 р.	10 Гб
2	План Б	400 р.	20 Гб
3	План С	500 р.	30 Гб
4	План Д	600 р.	40 Гб

Result Table

AbonentID	AbonentName	TariffID	TariffID	TariffName	Cost	IncludedData
1	Анна	2	2	План Б	400 р.	20 Гб
4	Иван	3	3	План С	500 р.	30 Гб
2	Мария	1	1	План А	300 р.	10 Гб
NULL	NULL	NULL	4	План Д	600 р.	40Гб

Если тарифный план не имеет подключенных абонентов, в соответствующих столбцах таблицы «Абоненты» будут возвращены значения NULL.

FULL JOIN возвращает обе таблицы, объединенные в одну. Результаты, возвращаемые этим типом соединения, включают все строки из обеих таблиц. Там, где встречаются совпадения, значения связаны. Если соответствия из любой таблицы нет, вместо этого возвращается NULL. Например, в базе данных имеется таблица «Студент» и таблица «Курс». Каждый студент может быть записан на несколько курсов, и каждый курс может иметь несколько студентов. Третья таблица «Зачисление» (Enrollments) является связующей и содержит сведения о том, кто из студентов уже записался на один или несколько курсов. После применения FULL JOIN между таблицами «Студент» и «Зачисление», а также между таблицами «Курс» и «Зачисление» результирующая таблица будет содержать сведения обо всех студентах и всех курсах, даже если они не имеют соответствующих записей в связующей таблице записи на курс.

```
SELECT Student.Name, Course.CourseName
FROM Student
FULL JOIN Enrollment ON Student.StudentID = Enrollment.StudentID
FULL JOIN Course ON Enrollment.CourseID = Course.CourseID;
```

Table "Student"

StudentID	Name
1	Анна
2	Иван
3	Мария

Table "Course"

CourseID	CourseName
101	История
102	Физика
103	Химия

Table "Enrollment"

StudentID	CourseID
1	101
1	102
2	102

Result Table

StudentID	Name	CourseID	Course Name
1	Анна	101	История
4	Анна	102	Физика
2	Иван	102	Физика
3	Мария	NULL	NULL
NUL	NULL	103	Химия

Таким образом, FULL JOIN позволяет посмотреть – кто из студентов еще не записался ни на один курс, и на какие курсы еще не записался ни один студент. FULL JOIN как раз и полезен для анализа данных с возможными пропущенными соответствиями, отладки запросов и проверки корректности данных, интеграции данных из нескольких источников.

LEFT JOIN (IF NULL) работает подобно LEFT JOIN, но он предоставляет дополнительную возможность для работы с NULL значениями. LEFT JOIN (IF NULL) обеспечивает включение всех строк из левой таблицы в результирующий набор, а также позволяет определить, как обрабатывать NULL значения, которые могут появиться в полях, полученных в результате объединения. Например, нужно объединить две таблицы - Table_A и Table_B их по колонке id и заменить NULL значения в Column2 из Table_B на строку 'Нет данных', если они встречаются:

```
SELECT Table_A.id, IFNULL(Table_B.Column2, 'Нет данных') AS Column2
FROM Table_A
LEFT JOIN Table_B ON Table_A.id = Table_B.id;
```

Также можно выполнить замену NULL значений на определенное значение с помощью функций COALESCE(), которая позволяет указать несколько аргументов и возвращает первое непустое значение из списка, в то время как функция IFNULL просто заменяет NULL на альтернативное значение. Например:

```
SELECT COALESCE(Column1, Column2, 'Значение по умолчанию') AS new_column
FROM table_name;
```

RIGHT JOIN с NULL позволяет управлять обработкой NULL-значений в результирующем наборе аналогично LEFT JOIN (IF NULL), только с учетом правой таблицы

CROSS JOIN – это своеобразный вариант соединения, который используется не так часто, как вышеперечисленные способы объединения, но важен для понимания. Он возвращает *декартово произведение*, т.е. комбинацию строк из обеих таблиц без каких-либо условий соединения между ними. В качестве примера соединение CROSS JOIN может использоваться при создании фильтров в интернет-магазине. Например, человек ищет обувь по характеристикам «тип» и «размер»:

```
SELECT TypesOfShoes.ShoeType, Sizes.ShoeSize
FROM TypesOfShoes
CROSS JOIN Sizes;
```

Table " TypesOfShoes"

ShoeType
Кроссовки
Туфли
Сапоги

Table "Size"

ShoeSize
36
37
38
39
40

Result Table "

ShoeType	ShoeSize
Кроссовки	36
Кроссовки	37
Кроссовки	38
Кроссовки	39
Кроссовки	40
Туфли	36
Туфли	37
Туфли	38

Результатом такого запроса должны быть все возможные комбинации типа с размером. Однако такое «картезианское произведение» может привести к огромному количеству строк в результирующей таблице. Если исходные таблицы содержат большое количество записей, а также, если не используются надлежащие фильтры или ограничения, результат CROSS JOIN может привести к экспоненциальному росту числа комбинаций строк.

SELF JOIN – это «самосоединение» или объединение строк внутри одной таблицы в том случае, когда нужно проанализировать зависимости внутри одной таблицы. Примеры использования SELF JOIN – обработка деревьев или графов, которые хранятся в одной таблице – генеалогические деревья, иерархии в компаниях, сложные структуры данных, такие как сетевой маркетинг. Механизм SELF JOIN полезен и при поиске дубликатов, поскольку позволяет сопоставить записи в одной таблице для определения повторяющихся данных. Чтобы сформировать запрос, следует указать одну и ту же таблицу дважды под разными именами – псевдонимами. В результате оператор «воспринимает» переданные ему сущности как разные. Этот вариант объединения может быть и внутренним, и внешним. Его особенность заключается в том, что таблица при таком режиме присоединяется сама к себе. Рассмотрим пример таблицы «Сотрудники», хранящей иерархию компании. Каждая строка представляет собой запись о сотруднике. У каждого сотрудника есть уникальный идентификатор EmployeeID, имя Employee_Name и, если он является подчиненным, идентификатор его менеджера ManagerID. В запросе реализуется SELF JOIN таблицы «Employees» с использованием ***псевдонимов*** «e1» и «e2». Таблица соединяется сама с собой по условию равенства идентификатора менеджера в первой таблице идентификатору сотрудника во второй таблице.

```
SELECT e1.EmployeeName AS Employee, e2.EmployeeName AS Manager
FROM Employee e1
JOIN Employee e2 ON e1.ManagerID = e2.EmployeeID;
```

Table "Employee"

EmployeeID	EmployeeName	ManagerID
1	Мария	2
2	Иван	4
3	Анна	2
4	Евгений	NULL

Result Table

Employee	Manager
Мария	Иван
Иван	Евгений
Анна	Иван

После выполнения запроса будут получены имена сотрудников и их менеджеров. Слово SELF в теле запроса указывать не нужно. Однако необходимо указать имена столбцов результирующей таблицы. Названия столбцов должны отражать смысл их содержимого, чтобы обеспечить понимание структуры данных.

JOIN – это мощный инструмент в SQL, который позволяет получать более сложные и полные результаты запросов, объединяя информацию из различных источников данных.

Понимание JOIN позволяет эффективно работать с данными в базе данных, выполнять сложные запросы и получать нужную информацию.

Знание JOIN также помогает оптимизировать структуру базы данных, улучшить производительность запросов и создавать более гибкие и мощные приложения, основанные на базе данных.

Алиасы (псевдоимена)

Алиасы в запросах используются для улучшения читаемости и понятности, особенно когда работают с более сложными запросами. Чтобы понять, почему они нужны, рассмотрим несколько примеров.

1. *Упрощение записи*: Когда в запросах используются несколько таблиц или вложенные подзапросы, алиасы позволяют сократить имена таблиц или колонок, делая запрос более компактным. Например, если таблица называется `Author`, можно использовать псевдоним `a`, чтобы работать с ним быстрее и без излишних повторений:

Без алиасов	С алиасами

`SELECT Author.Name, Book.Title`
`FROM Author`
`JOIN Book ON Author.AuthorID =`
`Book.Author`
	`SELECT a.Name, b.Title`
	`FROM Author a`
	`JOIN Book b ON a.AuthorID = b.Author`

Здесь это не особо видно, но читаемость сильно выигрывает, когда вы работаете с большим количеством таблиц и условий.

2. *Понимание контекста*: В случае сложных запросов с вложенными подзапросами алиасы дают ясное представление о том, какой элемент относится к какой таблице. Например, в запросе с вложенным подзапросом:

```
SELECT b.Title,
       (SELECT a.Name
        FROM Author a
        WHERE a.AuthorID = b.Author)
FROM Book b;
```

Здесь алиас `a` для таблицы `Author` в подзапросе помогает быстро понять, что `a.Name` относится к имени автора, а `b.Title` — к названию книги. Без алиасов подзапрос станет менее очевидным и трудным для восприятия, особенно если таблиц больше.

3. *Уникальность идентификаторов*: Когда в запросе участвуют одинаковые столбцы из разных таблиц, алиасы помогают избежать путаницы. Например, если в запросе участвуют несколько таблиц с полем `Name`, алиасы позволяют явно указать, из какой таблицы это поле берется.

В итоге алиасы делают код проще для восприятия и удобнее для дальнейшей работы. Это особенно важно в крупных проектах, когда запросы становятся сложными, и нужно поддерживать чистоту и понятность кода.

Вложенные запросы

Вложенные запросы (subqueries) – это SQL-запросы внутри другого SQL-запроса.
1. Вложенные запросы могут быть использованы для того, чтобы сначала выбрать одно значение в вложенном запросе, а затем использовать это значение в внешнем запросе для выполнения дальнейших операций.

Пример: Найти название книги и имя ее автора без `JOIN`. Причем у книги только один автор.

```
SELECT Author.Name , Book.Title
    FROM Book
    JOIN Author ON Author.AuthorID = Book.Author
```

```
SELECT Title,
(SELECT Name FROM Author WHERE AuthorID = Book.Author)
FROM Book;
```

2. Если вложенный запрос возвращает несколько строк, а внешний запрос ожидает одно значение (например, при использовании оператора =), это может вызвать ошибку или некорректный результат. Поэтому следует использовать оператор IN.

Пример с оператором IN. Причем у книги может быть несколько авторов.

```
SELECT Title
FROM Book
WHERE Author IN (SELECT Author FROM Book WHERE Title = 'Мать');
```

В этом случае подзапрос возвращает несколько авторов, и оператор IN корректно сопоставит их с полем Author, не вызывая ошибок.

3. Вложенные запросы могут помочь если связи между таблицами сложные или JOIN приводит к дублированию строк,

Пример: Найти книги, у которых год публикации раньше самого старого года публикации в таблице. JOIN тут не поможет, потому что мы сравниваем с одной агрегированной величиной.

```
SELECT Title, PublicationYear
FROM Book
WHERE PublicationYear < (SELECT MIN(PublicationYear) FROM Book);
```

Вложенные запросы помогают отфильтровать данные. Это удобно, когда мы фильтруем таблицу на основе условий из другой таблицы.

Пример: Найти все книги, написанные авторами, у которых есть хотя бы одна книга после 2000 года.

```
SELECT Title
FROM Book
WHERE Author IN (SELECT AuthorID FROM Book WHERE PublicationYear > 2000);
```

В большинстве случаев JOIN выполняются быстрее, чем подзапросы. Это связано с тем, что в JOIN СУБД может создать план выполнения, который лучше подходит для запроса и предсказывает, какие данные следует загрузить для обработки. В отличие от подзапроса, где она будет выполнять все запросы и загружать все их данные для выполнения обработки.

Важно помнить, что использование вложенных запросов может стать менее эффективным, если база данных становится очень большой и запросы становятся сложными. В таких случаях использование JOIN может быть быстрее, так как соединение данных может быть выполнено за один проход, а не через несколько запросов.

Итак: JOIN предпочтительнее, если нужно объединить таблицы, а вложенные запросы полезны, когда нужно вернуть одно значение для фильтрации или сравнения.