9. Теория БД и SQL: Нормальные формы, JOIN, Оптимизация

1. Нормальные формы (1NF, 2NF, 3NF)

Нормализация БД — процесс организации данных для уменьшения избыточности и улучшения целостности.

1NF (Первая нормальная форма)

- Каждая ячейка содержит **атомарное значение** (не списки, не множества).
- Нет повторяющихся строк.
- Определен **первичный ключ**.

❌ **Плохо** (нарушение 1NF):

student_id	courses
1	Math, Physics

✅ **Хорошо** (1NF):

student_id	course
1	Math
1	Physics

2NF (Вторая нормальная форма)

- Должна быть **1NF**.
- Все неключевые атрибуты **полностью зависят от первичного ключа** (нет частичной зависимости).

❌ **Плохо** (частичная зависимость):

order_id (PK)	product_id	product_name	quantity
1	101	Laptop	2

✅ **Хорошо** (2NF):
Таблица `orders`:

order_id (PK)	product_id	quantity

Таблица `products`:

product_id (PK)	product_name

3NF (Третья нормальная форма)

- Должна быть **2NF**.
- Нет **транзитивных зависимостей** (неключевые атрибуты не зависят от других неключевых атрибутов).

❌ **Плохо** (транзитивная зависимость):

employee_id (PK)	department	manager
1	IT	John Doe

✅ **Хорошо** (3NF):
Таблица `employees`:

employee_id (PK)	department_id

Таблица `departments`:

department_id (PK)	department	manager

---

2. Типы JOIN

Операции для объединения данных из нескольких таблиц.

INNER JOIN

- Возвращает **только совпадающие строки** из обеих таблиц.

```
SELECT u.name, o.order_id
FROM users u
INNER JOIN orders o ON u.id = o.user_id;
```

(Если у пользователя нет заказов, он не попадет в результат.)

LEFT JOIN (LEFT OUTER JOIN)

- Возвращает **все строки из левой таблицы** + совпадения справа.
- Если справа нет совпадений — `NULL`.

```
SELECT u.name, o.order_id
FROM users u
LEFT JOIN orders o ON u.id = o.user_id;
```

(Покажет всех пользователей, даже без заказов.)

RIGHT JOIN (RIGHT OUTER JOIN)

- Аналогично `LEFT JOIN`, но **все строки из правой таблицы**.

```
SELECT u.name, o.order_id
FROM users u
RIGHT JOIN orders o ON u.id = o.user_id;
```

(Покажет все заказы, даже если пользователь удален.)

FULL JOIN (FULL OUTER JOIN)

- Возвращает **все строки из обеих таблиц**, заполняя `NULL` при отсутствии совпадений.

```
SELECT u.name, o.order_id
FROM users u
FULL JOIN orders o ON u.id = o.user_id;
```

(Покажет и пользователей без заказов, и заказы без пользователей.)

---

3. Оптимизация SQL-запросов

Индексы

- Ускоряют поиск (`WHERE`, `JOIN`, `ORDER BY`).
- Замедляют `INSERT`/`UPDATE`/`DELETE` (индекс нужно перестраивать).
- Пример создания:

```
CREATE INDEX idx_user_email ON users(email);
```

EXPLAIN

Показывает **план выполнения запроса**, помогая найти узкие места.

```
EXPLAIN SELECT * FROM users WHERE age > 25;
```

Ключевые метрики:

- `Seq Scan` (полный перебор строк) → медленно, нужен индекс.
- `Index Scan` (использование индекса) → хорошо.
- `Cost` – оценка затратности операции.

Как улучшить запрос?

1. Добавить индексы на часто используемые поля.
2. Избегать `SELECT *` – выбирать только нужные столбцы.
3. Использовать `LIMIT` для больших выборок.
4. Оптимизировать `JOIN` (проверять порядок таблиц).

Вывод

- **Нормальные формы** уменьшают дублирование и аномалии.
- **JOIN** позволяет гибко комбинировать данные.
- **Индексы + EXPLAIN** – ключевые инструменты оптимизации.

Правильное проектирование БД и запросов сильно влияет на производительность! 🚀