Содержание

Часть I
Реляционная модель данных

1. Основные функции СУБД

Определение 1. *База данных* — это упорядоченный набор структурированной информации или данных которые обычно хранятся в электронном виде в компьютерной системе.

База данных обычно управляется *системой управления базами данных (СУБД).*
Функции СУБД:

- средства постоянного хранения данных;
- поддержка безопасности данных и защита от несанкционированного доступа;
- обеспечение согласованности данных;
- поддержка высокоуровневых эффективных языков запросов.

2. Модель данных сущность—связь: сущности, атрибуты и множества сущностей

Основные элементы диаграммы "сущность—связь":

- сущность (прямоугольник);
- атрибут (овал);
- связь (ромб).

Определение 2. *Сущность* — это абстрактный объект определённого вида. Любой предмет или понятие, информацию о котором мы будем хранить.

У каждой должно быть уникальное имя. К одному и тому же имени всегда должна применяться одна и та же интерпретация.

Определение 3. Набор экземпляров сущностей образует *множество.*

Отдельные характеристики объекта называются *атрибутами.*
Требования к атрибутам:

- каждый атрибут имеет *имя* и *тип данных*;
- сущность может обладать любым количеством атрибутов;
- значение атрибута атомарно;
- сущность и её атрибуты на диаграмме соединяются ненаправленными дугами;
- значения атрибутов выбираются из соответствующего множества значений.

Определение 4. *Атрибут* — именованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей. Атрибуты используются определения того, какая информация должна быть собрана о сущности.

Определение 5. *Ключ* — это один или несколько атрибутов объекта, по которому объект можно однозначно идентифицировать.

3. Модель данных сущность—связь: связи. Использование сущностей и связей при проектировании БД

Определение 6. *Связь* — это отношение между объектами.

В любой связи выделяются два конца, на каждом из которых указывается:

- имя конца связи;
- степень конца связи (сколько экземпляров данной сущности связывается).

Свойства.

1. Связи могут иметь собственные атрибуты.
2. Подобные связи объединяются в множества.
3. Связи не могут существовать без связываемых сущностей.
4. Ключ связи включает ключи связываемых сущностей и, возможно, выделенные атрибуты связи.

Характеристики связи:

- Размерность:
  - бинарные;
  - тернарные;
  - n-арные;
  - рекурсивные.
- Мощность:
  - 1:1;
  - 1:N;
  - M:N.
- Модальность:
  - обязательные ("должен") — экземпляр одной сущности **обязан** быть связан не менее чем с одним экземпляром другой сущности;
  - необязательные (условные, "может") — экземпляр одной сущности **может быть** связан с одним или несколькими экземплярами другой сущности, а может быть и не связан ни с одним экземпляром.

Шаги при создании ERD:

1. определить сущности;
2. определить атрибуты сущностей;
3. определить первичные ключи;
4. определить отношения между сущностями;
5. определить мощность связей;
6. нарисовать ERD;
7. проверить ERD.

4. Реляционная модель данных: отношения, таблицы, домены, атрибуты. Описание таблиц в языке SQL

Определение 7. *Реляционная модель данных* — логическая модель данных, описывающая:

- структуры данных в виде наборов отношений;
- теоретико-множественные операции над данными: объединение, пересечение, разность и декартово произведение;
- специальные реляционные операции: селекция, проекция, соединение и деление;
- специальные правила, обеспечивающие целостность данных.

Структуры данных:

- *домены* — множества, элементы которых рассматриваются как скалярные значения;

- *отношения* — предикаты, заданные на прямом произведении доменов;
- *атрибуты* — заголовок отношения; количество атрибутов — *размерность* отношения.

Атрибуты: $A_1, A_2, \ldots, A_n$.
Домены: $D_1, D_2, \ldots, D_n$.
Кортежи: $t = \langle a_1, a_2, \ldots, a_n \rangle$, $a_i \in D_i$; *Отношения*: $R \subset D_1 \times D_2 \times \cdots \times D_n$.

Замечание. Кортежи отличаются друг от друга только значением своих атрибутов. В реляционной БД не может быть двух одинаковых кортежей в одной таблице.

Определение 8. *Таблицы* — это единицы хранения данных в базе. Таблицы хранят все данные, к которым может обращаться пользователь. Реляционную БД можно рассматривать как коллекцию простых таблиц, связанных между собой.

Таблицы в БД создаются с помощью оператора `CREATE`:

```
CREATE TABLE Table_Name(
        {{имя столбца} {тип данных} [значение по-умолчанию] [правила целостности]}+)
```

5. Ограничения целостности: уникальность атрибута, нулевые значения, значения по-умолчанию

Определение 9. *Правила целостности* — это правила (ограничения), которым должны соответствовать данные. Их используют для того, чтобы обеспечить:

- связи между сущностями (*ссылочная целостность*);
- ограничения по значениям атрибутов в сущностях (*сущностная целостность*).

Уникальность значения:

```
CREATE TABLE people (id INT UNIQUE NULL)
```

Несколько полей (или их комбинаций) могут быть уникальны. Уникальное значение может быть `NULL`.

6. Ограничения целостности: первичный ключ

```
CREATE TABLE people (id INT PRIMARY KEY)
```

`PRIMARY KEY` **всегда** `NOT NULL`. `PRIMARY KEY` только один в таблице.

Первичный ключ может состоять из нескольких атрибутов:

```
CREATE TABLE people (name CHAR(20), address VARCHAR(35), PRIMARY KEY(name, address))
```

7. Ограничения целостности: внешний ключ

```
CREATE TABLE phone (id_p INT REFERENCES people (id))
```

или

```
CREATE TABLE phone (id_p INT, FOREIGN KEY (id_p) REFERENCES people (id))
```

Требования к внешнему ключу:

- поле, на которое ссылается внешний ключ, должно быть `PRIMARY KEY` или `UNIQUE`;
- оба поля должны быть строго одного типа.

8. Ограничения целостности: ограничение на значения атрибута

```
CREATE TABLE people (id INT PRMARY KEY, gender CHAR CHECK (gender IN ('F', 'M')))
CREATE TABLE people (id INT PRIMARY KEY, birthday DATE, beg_date DATE,
        CHECK (birthday < beg_date))
CREATE TABLE people (id INT PRIMARY KEY, gender CHAR,
        CONSTRAINT chk_Person CHECK (gender IN ('F', 'M')))
```

9. Ограничения целостности: вычисляемые атрибуты

```
CREATE TABLE people (id INT PRIMARY KEY, salary INT, tax AS salary * 0.13)
```

Расширение PostgreSQL:

```
CREATE TABLE people (id INT PRIMARY KEY, salary INT,
        tax INT GENERATED ALWAYS AS (salary * 0.13) STORED)
```

10. Реляционная модель данных: алгебраические операции

Все операции *реляционной алгебры* производятся над отношением, и результатом операции является отношение.

Реляционная алгебра является *замкнутой*: в качестве аргументов в реляционные операторы можно подставлять другие реляционные операторы, подходящие по типу. В реляционных выражениях можно использовать вложенные выражения сколь угодно вложенной структуры.

Каждое отношение обязано иметь уникальное имя в пределах БД. Если отношения подставляются в качестве аргументов в другие реляционные выражения, то они могут быть неименованными.

Операции реляционной алгебры делятся на два типа:

- теоретико-множественные операции;
- специальные реляционные операции.

Теоретико-множественные операции:

- объединение отношений;
- пересечение отношений;
- разность отношений;
- декартово произведение отношений.

Два отношения *совместимы по взятию декартова произведения* в том и только в том случае, если пересечение имён атрибутов, взятых из схем отношений, пусто. Любые два отношения могут стать совместимыми по взятию декартова произведения, если применить к одному из них операцию переименования.

Свойства операций:

- ассоциативность (кроме разности);
- коммутативность (кроме разности).

11. Реляционная модель данных: реляционные операции

Специальные реляционные операции:

- *ограничение отношения (селекция)* — горизонтальная вырезка;
- *проекция отношения* — вертикальная вырезка;
- *соединение отношений* (по условию, эквисоединение, естественное соединение);
- *деление отношений*.

Селекция

```
R WHERE f
```

$$\sigma_f(R)$$

Условие ограничения имеет вид:

- (a `операция_сравнения` b), где a и b — имена атрибутов ограничиваемого отношения; атрибуты a и b определены на одном домене, для значений которого поддерживается операция сравнения;

- (a `операция_сравнения` const), где a — имя атрибута ограничиваемого отношения; атрибут a должен быть определён на домене или базовом типе, для значений которого поддерживается операция сравнения.

Условие может состоять из нескольких простых логических выражений, связанных булевскими операторами `AND, NOT, OR`.

Результатом селекции является отношение, заголовок которого совпадает с заголовком отношения-операнда, а в тело входят те кортежи отношения-операнда, для которых условие ограничения выполнено.

Проекция

Определение 10. *Проекцией* отношения R по атрибутам $X, Y, \ldots, Z$ где каждый из атрибутов принадлежит отношению, называется отношение с заголовком $(X, Y, \ldots, Z)$ и телом, содержащим множество кортежей вида $(x, y, \ldots, z)$ таких, для которых найдутся кортежи со значением атрибута X равным x, ..., значением атрибута Z равным z.

$$\pi_{(X,Y,\ldots,Z)}(R) = \{\, x, y, \ldots, z : \quad \exists a_1, a_2, \ldots, a_n \in R \;\&\; x = a_{i1},\ y = a_{i2}, \ldots, z = a_{im} \,\}$$

Соединение по условию (θ-соединение)

Три операнда: соединяемые отношения и условие. Операнды должны быть совместимы по взятию декартова произведения.

```
A JOIN B WHERE f = (A x B) WHERE f
```

$$R \rhd\lhd_f S = \sigma_f(R \times S)$$

Эквисоединение

Определение 11. Операция соединения называется операцией *эквисоединения* (`EQUI JOIN`), если условие соединения имеет вид $(a = b)$.

Если соединение происходит по атрибутам с одинаковыми именами, то в результирующем отношении появляется два атрибута с одинаковыми значениями. В таком случае нужно брать проекцию по всем атрибутам, кроме одного из дублирующих.

Естественное соединение

Операция *естественного соединения* применяется к паре отношений $R(A, X)$ и $S(X, B)$, обладающих общим атрибутом X.

$$T(A, X, B), \quad A \rhd\lhd B$$

В синтаксисе естественного соединения не указывается, по каким атрибутам производится соединение. Естественное соединение производится **по всем** одинаковым атрибутам.

$$R \rhd\lhd S = \pi_{\text{атрибуты } R,\ S \setminus S.A} \sigma_{R.A=S.A}(R \times S)$$

Деление

Определение 12. Результатом *деления* A на B является "унарное" отношение $C(a)$, тело которого состоит из кортежей v таких, что в теле отношения A содержатся кортежи $\langle v, w \rangle$ для любого w из B.

$$(A \texttt{ DIVIDE BY } B) = C : \quad C \times B \subset A$$

$$R \texttt{ DIVIDE BY } S = \pi_{1,2,\ldots,r-s}(R) - \pi_{1,2,\ldots,r-s}\big(\pi_{1,2,\ldots,r-s}(R \times S) \setminus R\big)$$

Свойства.

1. Коммутативность
   - для декартова произведения:
   $$R_1 \times R_2 = R_2 \times R_1$$
   - для соединений:
   $$R_1 \rhd\lhd_f R_2 = R_2 \rhd\lhd_f R_1$$
2. Ассоциативность:
   - для декартова произведения:
   $$(R_1 \times R_2) \times R_3 = R_1 \times (R_2 \times R_3)$$
   - для соединений:
   $$(R_1 \rhd\lhd_{f_1} R_2) \rhd\lhd_{f_2} R_3 = R_1 \rhd\lhd_{f_1} (R_2 \rhd\lhd_{f_2} R_3)$$
3. Комбинация (каскад):
   - для селекций:
   $$\sigma_{f_1}\big(\sigma_{f_2}(R)\big) = \sigma_{f_1 \vee f_2}(R)$$
   - для проекций:
   $$\pi_{A_1,A_2,\ldots,A_m}\big(\pi_{B_1,B_2,\ldots,B_n}(R)\big) = \pi_{A_1,A_2,\ldots,A_m}(R), \quad \text{где } \{ A_m \} \subset \{ B_n \}$$
4. Перестановка
   - селекции и проекции:
   $$\sigma_f \pi_{A_1,A_2,\ldots,A_m}(R) = \pi_{A_1,A_2,\ldots,A_m} \sigma_f(R)$$
   - селекции и объединения:
   $$\sigma_f(R_1 \cup R_2) = \sigma_f(R_1) \cup \sigma_f(R_2)$$
   - селекции и декартова произведения:
   $$\sigma_f(R_1 \times R_2) = \big(\sigma_{f_1}(R_1)\big) \times \big(\sigma_{f_2}(R_2)\big)$$
   - селекции и разности:
   $$\sigma_f(R_1 \setminus R_2) = \sigma_f(R_1) \setminus \sigma_f(R_2)$$
   - селекции и пересечения:
   $$\sigma_f(R_1 \cap R_2) = \sigma_f(R_1) \cap \sigma_f(R_2)$$
   - проекции и декартова произведения:
   $$\pi_{A_1,A_2,\ldots,A_m}(R_1 \times R_2) = \big(\pi_{B_1,B_2,\ldots,B_n}(R_1)\big) \times \big(\pi_{C_1,C_2,\ldots,C_r}(R_2)\big)$$

12. Отображение модели сущность—связь в реляционную. Представление объектов

Каждая сущность превращается в таблицу. Имя сущности становится именем таблицы.

Каждый атрибут становится столбцом. Столбцы для необязательных атрибутов могут содержать неопределённые значения, столбцы для обязательных — не могут.

Компоненты уникального идентификатора сущности превращаются в ключ таблицы.

13. Отображение модели сущность—связь в реляционную. Представление связей

Связи также хранятся в отношении. Схема данного отношения составляется из ключевых атрибутов, участвующих в связи.

14. Функциональные зависимости и аномалии вставки, обновления, удаления

Определение 13 (функциональная зависимость). $R\{A_1, A_2, \ldots, A_n\}, \quad X, Y \subset \{A_1, A_2, \ldots, A_n\}$
$X \to Y$, если любому значению X соответствует ровно одно значение Y.
X называется *детерминантом*, Y — *зависимой частью*.

$$X \to Y \iff \left|\pi_Y\left(\sigma_{X=x}(R)\right)\right| \leq 1$$

Пусть $\{A_1, A_2, \ldots, A_n\} \to \{B_1, B, \ldots, B_m\}$. Функциональные зависимости:

- тривиальные:
$$\{B_1, B_2, \ldots, B_m\} \subset \{A_1, A_2, \ldots, A_n\}$$
- нетривиальные:
$$\{B_1, B_2, \ldots, B_m\} \not\subset \{A_1, A_2, \ldots, A_n\}, \quad \{A_1, A_2, \ldots, A_n\} \cap \{B_1, B_2, \ldots, B_m\} \neq \emptyset$$
- полностью нетривиальные:
$$\{A_1, A_2, \ldots, A_n\} \cap \{B_1, B_2, \ldots, B_m\} = \emptyset$$

Аксиомы Армстронга:

- рефлексивность:
$$B \subset A \implies A \to B$$
- пополнение:
$$A \to B \implies AC \to BC$$
- транзитивность:
$$A \to B,\ B \to C \implies A \to C$$

Из аксиом Армстронга можно получить правила вывода:

- объединение:
$$X \to Y,\ X \to Z \implies X \to YZ$$
- псевдотранзитивность:
$$X \to Y,\ WY \to Z \implies WX \to Z$$
- декомпозиция:
$$X \to Y,\ Z \subseteq Y \implies X \to Z$$

15. Нормализация: декомпозиция отношений. Нормальные формы

Определение 14. *Ключ* — минимальный набор атрибутов, который функционально определяет все остальные.

Определение 15. Y *полностью функционально зависит* от X, если Y функционально зависит от всех атрибутов, входящих в состав X, а не от какой-то его части.

Определение 16. Функциональная зависимость $A \to C$ называется *транзитивной*, если существует такой атрибут B, что $A \to B$ и $B \to C$ и отсутствует функциональная зависимость $C \to A$.

Определение 17. *Декомпозиция* — это разбиение на множества, может быть пересекающиеся, такие, что их объединение — это исходное отношение.

Восстановить исходное отношение можно только естественным соединением.
Нормальные формы:

- Первая:
  Значение каждого атрибута в таблице должно быть атомарно.
- Вторая:
  Таблица находится в 2НФ, если она находится в 1НФ, и каждый атрибут полностью зависит от любого его ключа, но не от подмножества ключа.
- Третья:
  Отношение находится в 2НФ, и любой атрибут, не являющийся первичным, нетранзитивно зависит от любого возможного ключа.
- Бойса—Кодда:
  Если $X \to A,\ A \notin X$, то $X \supseteq$ ключ R.

16. Нормализация: многозначные зависимости

Определение 18. Пусть A и B — два атрибута отношения R.
Между этими атрибутами существует *многозначная* зависимость, если значению a атрибута A соответствует множество значений $\{ b_1, b_2, \ldots, b_k \}$ атрибута B.

Обозначение. $A \twoheadrightarrow B$

Лемма 1 (Фейджина). В отношении $R\{ A, B, C \}$ выполняется MVD $A \twoheadrightarrow B$ в том и только в том случае, когда выполняется MVD $A \twoheadrightarrow C$.

Чтобы подчеркнуть этот факт, иногда пишут $A \twoheadrightarrow B|C$.

Теорема 1 (Фейджина). $R\{ A, B, C \}$
Переменная-отношение R будет равна соединению её проекций $[A, B]$ и $[A, C]$ тогда и только тогда, когда для переменной-отношения R выполняется многозначная зависимость $A \twoheadrightarrow B|C$.

Четвёртая нормальная форма:
Отношение R находится в 4НФ, если оно находится в НФБК, и детерминант любой нетривиальной MVD является ключом R.

Часть II
Язык SQL

Примечание. В этой части предполагается, что читатель в состоянии самостоятельно сходу написать несколько десятков запросов на каждую конструкцию. В презентации было слайдов 150 только на примеры.

17. Язык запросов SQL: операции реляционной алгебры

Проекция:

```
SELECT name FROM person
```

Селекция:

```
SELECT name, price FROM items WHERE price >= 100 AND price <= 150
```

Декартово произведение:

```
SELECT * FROM person, dept
```

Соединение:

- Внутреннее:

```
SELECT ... FROM t_1, t_2 WHERE t_1.f_1 = t_2.f_2
SELECT ... FROM t_1 [INNER] JOIN t_2 ON t_1.f_1 = t_2.f_2
```

- Внешнее:
  - `JOIN`: строки, в которых есть хотя бы одно совпадение в обеих таблицах;
  - `LEFT JOIN`: строки из левой таблицы, даже если нет совпадения в правой;
  - `RIGHT JOIN`: строки из правой таблицы, даже если нет соединения в левой;
  - `FULL JOIN`: строки из обеих таблиц.
- Естественное:

```
SELECT group, st FROM S JOIN R ON S.lang = R.lang
SELECT * FROM S JOIN R USING (lang)
SELECT * FROM S NATURAL JOIN
```

Теоретико-множественные операции:

- `UNION`;
- `INTERSECT`;
- `EXCEPT`.

Отношения должны быть совместимы, т. е. иметь одинаковое количество полей с совместимыми типами данных. По-умолчанию все операции `DISTINCT`.

18. Язык запросов SQL: вложенные подзапросы

Определение 19. *Вложенный SQL-запрос* — это отдельный запрос, который используется внутри SQL инструкции.

Инструкцию, в которой используется вложенный запрос, называют *внешним SQL-запросом*.

Вложенные запросы:

- `SELECT`:

```
SELECT *, (SELECT name FROM car_owner WHERE inn = owner) FROM car
```

- `FROM`:

```
SELECT region FROM (SELECT * FROM car WHERE ...)
```

- `WHERE`:

```
SELECT mark FROM T WHERE mark = (SELECT MAX(mark) FROM T)
```

19. Язык запросов SQL: агрегирование и упорядочение

Агрегатные функции:

- `COUNT` — количество записей в выходном наборе;
- `MIN, MAX` — наименьшее/наибольшее из множества значений;
- `AVG` — среднее значение;
- `SUM` — сумма множества значений.

Упорядочение:

```
SELECT name FROM person ORDER BY name [DESC | ASC] [NULLS {FIRST | LAST}]
```

20. Средства обновления данных в языке запросов SQL

Вставка:

```
INSERT INTO tbl (col1, col2) VALUES ('Data 1', 1), ('Data 2', 2), ...;
```

Изменение:

```
UPDATE tbl SET col1 = expr1[, col2 = expr2, ...] [WHERE condition];
```

Удаление:

```
DELETE FROM tbl [WHERE condition];
```

21. Типы данных SQL

Типы данных могут различаться в зависимости от реализации SQL. Простейшими типами данных в PostgreSQL можно считать:

- целочисленные значения;
- вещественные значения;
- строки фиксированной длины;
- строки переменной длины;
- дата и время.

Числовые типы:

Имя	Размер, байт	Описание
SMALLINT	2	целое в небольшом диапазоне
INTEGER	4	целое
BIGINT	8	целое в большом диапазоне
DECIMAL, NUMERIC	переменный	вещественное число с указанной точностью
REAL	4	вещественное число с переменной точностью
DOUBLE PRECISION	8	вещественное число с переменной точностью
SMALLSERIAL	2	небольшое целое с автоувеличением
SERIAL	4	целое с автоувеличением
BIGSERIAL	8	большое целое с автоувеличением

Строковые типы:

Имя	Описание
CHARACTER VARYING(n)	строка ограниченной переменной длины
CHARACTER(n)	строка фиксированной длины, дополненная пробелами
TEXT	строка неограниченной переменной длины

Типы даты и времени:

Имя	Размер, байт	Описание
TIMESTAMP [(p)] [WITHOUT TIME ZONE]	8	дата и время (без часового пояса)
TIMESTAMP [(p)] WITH TIME ZONE	8	дата и время (с часовым поясом)
DATE	4	дата
TIME [(p)] [WITHOUT TIME ZONE]	8	время суток
TIME [(p)] WITH TIME ZONE	12	время суток (с часовым поясом)
INTERVAL [поля] [(p)]	16	временной интервал

p — количество знаков после запятой в поле секунд (от 0 до 6).

Задание пользовательских типов:

```
CREATE TYPE mytype AS VARCHAR(11) NOT NULL
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy')
```

Явное преобразование типов:

```
SELECT CAST (10.6 AS INT)
SELECT CONVERT (INT, 10.6)
```

22. Временные таблицы и табличные переменные

Временные таблицы автоматически удалятся в конце текущей сессии.

```
CREATE TEMP TABLE tbl (id INT, name TEXT);
```

Таблицы могут быть переменными в процедурном расширении. Например, можно возвращать таблицу из функции:

```
CREATE FUNCTION get_table(a INTEGER)
        RETURNS TABLE (id INT, name TEXT)
        AS $$ BEGIN
                RETURN QUERY SELECT id, name FROM table1 WHERE id > a;
        END; $$ LANGUAGE plpgsql;
```

23. Представления (обычные, с фиксацией схемы, индексируемые)

Представления используются:

- для упрощения и настройки восприятия информации в БД каждым пользователем;
- в качестве механизма безопасности;
- для предоставления интерфейса обратной совместимости.

Создание представлений:

```
CREATE [OR REPLACE] VIEW citizens AS
        SELECT client_id, last_name, city FROM client WHERE client = 'Moscow'
```

Изменение данных в представлении возможно, если при его создании:

- не используется ключевое слово `DISTINCT`;
- данные извлекаются только из одной таблицы;
- в списке полей нет арифметических выражений;
- не используются подзапросы;
- не используется группирование;
- не используется `UNION`;

Создание материализованных представлений:

```
CREATE MATERIALIZED VIEW view_name AS select_stmt [WITH [NO] DATA]
```

`WITH NO DATA` — представление создаётся, но не заполняется данными.

Обновление данных:

```
REFRESH MATERIALIZED VIEW view_name
REFRESH MATERIALIZED VIEW CONCURRENTLY view_name
```

Во втором случае необходим уникальный индекс, построенный до обновления.

Создание временных представлений (создаются на действующую сессию):

```
CREATE TEMP VIEW view_name AS select_stmt
```

24. Обобщённые табличные выражения

> **Определение 20.** *Обобщённое табличное представление* представляет собой временно именованный результирующий набор (CTE), за которым следуют одиночные инструкции `SELECT`, `INSERT`, `UPDATE` или `DELETE`.

Структура CTE:

```
WITH expr_name (col_name[, ...]) AS (CTE_query_definition)
```

Рекурсивные обобщённые табличные выражения:

```
WITH RECURSIVE t(n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM t WHERE n < 15) SELECT n FROM t
WITH RECURSIVE t(n) AS (SELECT 1 UNION ALL SELECT n + 1 FROM t) SELECT n FROM t LIMIT 100
```

25. Хранимые процедуры в базе данных

Определение 21. *Хранимая процедура* — это набор операторов процедурного языка и языка SQL, который компилируется системой в единый "план исполнения".

Хранимые процедуры хранятся как объекты в базе. Они создаются и удаляются операторами `CREATE` и `DROP`.

```
CREATE [OR REPLACE] PROCEDURE my_sql_proc (a INT) LANGUAGE SQL AS
        $$ INSERT INTO tbl (field_1) VALUES (a) $$
CALL my_sql_proc (100)
```

26. Курсоры в базах данных

Определение 22. *Курсор* — это область в памяти базы данных, которая предназначена для хранения запроса SQL.

В памяти сохраняется и строка данных запроса, называемая *текущим значением*, или *текущей строкой* курсора. Указанная область в памяти поименована и доступна для прикладных программ.

Действия с курсорами:

- `DECLARE` — создание курсора;
- `OPEN` — открытие курсора, т. е. наполнение его данными;
- `FETCH` — выборка из курсора и изменение строк данных с его помощью;
- `CLOSE` — закрытие курсора;
- `DEALLOCATE` — освобождение курсора.

```
DECLARE c_name [[NO] SCROLL] CURSOR [(param)] FOR SELECT ...
DECLARE c_name REFCURSOR
```

В схеме со *статическим курсором* информация читается из БД один раз и хранится в виде моментального снимка. На время открытия курсора сервер устанавливает блокировку на все строки, включённые его полный результирующий набор. Статический курсор не изменяется после создания и всегда отображает тот набор данных, который существовал на момент его открытия.

```
DECLARE c_name INSENSITIVE [SCROLL] CURSOR FOR select_stmt
```

Динамические курсоры отражают все изменения строк в результирующем наборе при прокрутке курсора. Значения типа данных, порядок и членство строк в результирующем наборе могут меняться для каждой выборки. Обновления видны сразу, если они сделаны посредством курсора.

```
DECLARE c_name [SCROLL] CURSOR FOR select_stmt
        [FOR {READ_ONLY | UPDATE [OF col[, ...]]}]
```

Считать текущую строку в переменные:

```
DECLARE id INT, name VARCHAR;
FETCH FROM c_1 INTO id, name;
```

Прокручиваемый курсор позволяет использовать больше возможностей `FETCH`:

```
FETCH {NEXT | PRIOR | FIRST | LAST | ABSOLUTE n | RELATIVE n} FROM ...
```

Замечание. Курсор почти всегда использует дополнительные ресурсы сервера, что ведёт к резкому падению производительности.

27. Скалярные функции, определяемые пользователем

```
CREATE [OR REPLACE] FUNCTION name (params) RETURNS res_type {IMMUTABLE | STABLE | VOLATILE} AS
        $$ ... $$ LANGUAGE plpgsql
```

Функции:

- `VOLATILE` — может изменять данные в базе; может возвращать различные результаты с одинаковыми аргументами;
- `STABLE` — не может изменять данные в базе; возвращает одинаковый результат для всех вызовов с одинаковыми аргументами в одном операторе;
- `IMMUTABLE` — не может изменять данные в базе; всегда возвращает одинаковые результаты для одних и тех же аргументов.

```
CREATE FUNCTION fun_1 (a INT) RETURNS INT AS
        $$ BEGIN
                RETURN a + 1;
        END; $$ LANGUAGE plpgsql;
```

28. Табличные функции, определяемые пользователем

```
CREATE FUNCTION get_table (a INTEGER)
        RETURNS TABLE (
                new_id INT,
                new_name TEXT
        )
        AS $$ BEGIN
                RETURN QUERY SELECT id, name FROM table_1 WHERE id > a;
        END; $$ LANGUAGE plpgsql;
```

29. Активные базы данных: триггеры

Определение 23. *Триггер* — это откомпилированная SQL-процедура. Исполнение обусловлено наступлением определённых событий внутри базы.

Назначение триггеров:

- проверка корректности введённых данных и выполнение сложных ограничений целостности;
- накопление аудиторской информации;
- автоматическое оповещение других модулей об изменениях информации;
- реализация бизнес-правил;
- организация каскадных воздействий на таблицы БД;
- поддержка репликации.

```
CREATE TRIGGER tr BEFORE UPDATE ON table_1 FOR EACH ROW EXECUTE PROCEDURE pt()
```

При запуске триггера есть доступ к двум записям `NEW` и `OLD`.
Пример триггерной функции:

```
CREATE FUNCTION fun() RETURNS TRIGGER AS
        $$ BEGIN
                IF NEW.id IS NULL THEN
                        RAISE EXCEPTION 'id cannot be null';
                END IF;
                RETURN NEW;
        END; $$ LANGUAGE plpgsql;
```

Часть III
Индексы

30. Кластеризованные таблицы и индексы

Файлы конфигурации и файлы данных, используемые кластером базы данных, традиционно хранятся вместе в каталоге данных кластера, который обычно называют PGDATA.

Для каждой БД в кластере существует подкаталог внутри PGDATA/base, названный по OID базы в pg_database. Этот подкаталог по-умолчанию является местом хранения файлов базы данных.

Каждая таблица и индекс хранятся в отдельном файле. Для каждой таблицы и индекса есть карта свободного пространства. Когда объём таблицы или индекса превышает заданный размер, она делится на сегменты (ноды).

Основной единицей хранения данных является страница. Место на диске для размещения файла данных в БД логически разделяется на страницы с непрерывной нумерацией. Дисковые операции ввода-вывода осуществляются на уровне страницы. СУБД считывает или записывает целые страницы данных.

Кластерный индекс использует возможность физического индексирования данных. В результате будут отсортированы данные в самой таблице согласно порядку этого индекса. Добавление информации в таблицу приводит к изменению физического порядка данных. Кластерным может быть только один индекс в таблице.

Кластерный индекс обеспечивает самый быстрый поиск по заданному ключу.

31. Первичные и вторичные индексы. Плотные и неплотные индексы

На таблице можно определить до 249 некластерных индексов. В таком случае кластерный индекс становится *первичным*, а все некластерные — *вторичными*.

Если верить википедии, то плотный индекс ссылается на запись в таблице, а *неплотный* — на блок данных.

32. Индексы: B-дерево

- упорядочиваем таблицу по значению ключа;
- для каждого блока данных определяем минимальное значение ключа и адрес;
- эти пары также размещаем в блоках;
- наращиваем уровни, пока не появится уровень из одного блока.

33. Индексы: отфильтрованные, покрывающие, составной ключ

Покрывающие индексы

Определение 24. Если все столбцы запроса входят в состав ключа в индексе, то такой индекс называется *покрывающим*.

Покрывающие индексы могут повысить производительность запросов, так как данные, необходимые для удовлетворения требований запроса, присутствуют в самом индексе — не требует считывания страниц данных.

Отфильтрованные индексы

Определение 25. *Отфильтрованный индекс* — некластеризованный индекс, построенный по некоторому подмножеству значений ключа.

Может повысить производительность запросов, снизить затраты на обслуживание и хранение индексов по сравнению с полнотабличными индексами. Фильтрованные индексы полезны на больших таблицах.

Составной ключ

Индекс может быть создан на основании нескольких ключей. В состав одного составного индекса может входить до 16 столбцов. Все столбцы ключа составного индекса должны находится в одной таблице или одном и том же представлении.

34. Индексы: полнотекстовые

> **Определение 26.** *Полнотекстовый индекс* представляет собой словарь, в котором для каждого слова указано, где оно встречается.

Процесс построения полнотекстового индекса:

1. фильтрация;
2. разбиение по словам;
3. удаление стоп-слов;
4. иногда применяется *стемминг* (выделение основы слова) или *лемматизация* (приведение слова к нормальной форме);
5. преобразование конвертированных данных в инвертированный список слов;
6. заполнение полнотекстового индекса.

Заполнение индекса значениями:

- `MANUAL`

  ```
  ALTER FULLTEXT INDEX ON customers START FULL POPULATION
  ```

- `AUTO` (не обязательно мгновенно)

По-умолчанию индекс сопоставляется с системным стоп-листом.

```
ALTER FULLTEXT INDEX ON table_1 SET STOPLIST = sl
```

Запросы с полнотекстовым индексом используют предикаты `CONTAINS` и `FREETEXT` и функции `CONTAINSTABLE` и `FREETEXTTABLE`:

```
SELECT * FROM prod.prod_desc WHERE FREETEXT(desc, 'bike');
SELECT * FROM prod.prod_desc WHERE CONTAINS(desc, 'bike');
```

`CONTAINS` определяет точное или неточное совпадение, расстояние между словами или взвешенные совпадения. `FULLTEXT` используется для поиска значений, которые соответствуют условию по смыслу на основе тезауруса.

35. Индексы: хеширование

В памяти хранится стандартная хеш-таблица для ключевого столбца.

```
CREATE INDEX idx_name USING HASH ON t_name (col_name);
```

Недостатки хеш-индексов:

- хеш-таблица может быть слишком велика;
- если в один участок попало слишком много записей, придётся выделять дополнительный блок;
- неравномерность размещения записей, возникновение коллизий.

Тут я не согласен: на мой взгляд, это не проблемы, а задачи.

Хеш-индекс подходит для поиска только по равенству и не поддерживает транзакции. Хеш-индекс автоматически создаётся при `JOIN` при отсутствии других индексов.

36. Индексы: битовые шкалы

Для каждого значения индексируемого столбца в индекс на основе битовых карт входит одна строка, состоящая из значения столбца и битовой последовательности. Битовая последовательность имеет длину по количеству строк таблицы, в которой 1 означает, что в данной строке атрибут принимает заданное значение.

37. Индексы: R-дерево

R-дерево используется для геометрических и пространственных типов данных.

Для его построения каждая фигура окружается ограничивающим прямоугольником. Дальше принцип тот же, что и в B-дереве.

К недостаткам можно отнести то, что не всегда удаётся избежать перекрытий. В результате при поиске приходится просматривать несколько веток.

Возможные критерии разделения узла:

- минимальная площадь;
- минимальное перекрытие;
- минимальные границы.

Часть IV
Оптимизация запросов

38. Выполнение запросов: реализация операций реляционной алгебры

Операндами операций реляционной алгебры являются отношения. Для выполнения операций необходимо просмотреть все кортежи исходных отношений. Следствием этого является большая размерность операций РА. Уменьшения размерности можно достичь, изменяя последовательность выполняемых операций.

Оптимизация выполнения запросов реляционной алгебры основана на понятии *эквивалентности* реляционных выражений (см. вопрос 11).

Объединение выполняется путём сортировки данных для удаления одинаковых кортежей.

Виды соединения таблиц:

- Соединение с помощью вложенных циклов:
  - сложность — $O(N \log N)$;
  - используется, если хотя бы одна таблица достаточно маленькая, а бóльшая таблица имеет индекс по ключу соединения.
- Соединение слиянием:
  - сложность — $O(N + M)$;
  - используется, если обе таблицы отсортированы по столбцу слияния, а слияние происходит по равенству;
  - хорошо подходит для больших таблиц.
- Хеш-соединение:
  - сложность — $O(N \cdot h_c + M \cdot h_m + J)$;
  - использует таблицу хеширования и динамическую хеш-функцию для строк;
  - используется только в крайнем случае.

39. Задача оптимизации. Компоненты и функции оптимизатора запросов

Задача 1. По декларативной формулировке запроса требуется построить программу (*план выполнения запроса*), которая выполнялась бы максимально эффективно и выдавала бы результаты, соответствующие указанным в запросе свойствам. ✉

Таким образом, задача заключается в том, чтобы построить все возможные программы, дающие требуемый результат и выбрать из них такую, выполнение которой было бы наиболее эффективным.

Работа оптимизатора состоит из пяти стадий:

1. лексический и синтаксический анализ;
2. логическая оптимизация;
3. выбор альтернативных процедурных планов выполнения на основе информации, которой располагает оптимизатор;
4. по внутреннему представлению наиболее оптимального плана выполнения формируется процедурное представление;
5. выполнение запроса в соответствии с планом.

Два основных вида оптимизаторов:

- rule-based:
  - выбирает методы доступа на основе предположения о статичности СУБД;
  - учитывает иерархическое старшинство операций;
  - если для операции есть несколько путей выполнения, то выбирается путь с наименьшим (заранее заданным) рангом.
- cost-based;

40. Оптимизация запросов по стоимости. План выполнения запроса

Определение 27. *Стоимость (затраты)* — это оценка ожидаемого времени выполнения запроса с использованием конкретного плана выполнения.

Для каждого из выбранных планов оценивается предполагаемая стоимость выполнения запроса по этому плану. При оценках используется либо доступная оптимизатору статистическая информация о распределении данных, либо информация о механизмах реализации путей доступа.

Оптимизация выполнения запроса осуществляется в следующем порядке:

- вычисление выражений и условий, содержащих константы;
- преобразование сложной команды в эквивалентную ей с использованием соединения;
- если команда выполняется над представлением, то оптимизатор обычно объединяет запрос на создание представления и запрос к этому представлению;
- выбор метода оптимизации;
- выбор путей доступа к таблицам, к которым обращается запрос;
- выбор порядка соединения;
- выбор операции соединения для каждой команды соединения.

Часть V
Транзакции

41. Определение и основные свойства транзакций

Определение 28. *Транзакция* — это последовательность операций, производимых над базой данных и переводящих её из одного согласованного состояния в другое согласованное состояние.

Требования ACID:

- Atomicity:

  Каждая транзакция представляет собой единицу работы. Она не может быть разбита на меньшие части. Выполняются либо все действия, определённые в данной транзакции, либо не выполняется ни одно из них.

- Consistency:

  Гарантируется, что по мере выполнения транзакций данные переходят из одного согласованного состояния в другое — транзакция не разрушает взаимной согласованности данных. Для поддержания согласованности данных в процессе транзакции применяются правила, проверки, ограничения и триггеры.

- Isolation:

  Конкурирующие за доступ к БД транзакции физически обрабатываются изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.

- Durability:

  Если транзакция завершена успешно, то те изменения данных, которые ей произведены не могут быть потеряны ни при каких обстоятельствах.

Определение 29. База данных находится в *согласованном состоянии*, если для этого состояния выполнены все ограничения целостности.

Определение 30. *Ограничение целостности* — это некоторое утверждение, которое может быть истинным или ложным в зависимости от состояния базы данных.

42. Аномалии конкурентного выполнения транзакций

Аномалии конкурентного доступа:

- потерянное обновление (lost update):

```
   SELECT f2 FROM t1 WHERE f1 = 1;  |   SELECT f2 FROM t1 WHERE f1 = 1;
UPDATE t1 SET f2 = 20 WHERE f1 = 1; |
                                    | UPDATE tbl1 SET f2 = 25 WHERE f1 = 1;
```

- “грязное” чтение (dirty read):

```
    SELECT f2 FROM t1 WHERE f1 = 1;  |
UPDATE t1 SET f2 = f2 + 1 WHERE f1 = 1;
                                     | SELECT f2 FROM t1 WHERE f1 = 1;
            ROLLBACK WORK;           |
```

- неповторяющееся чтение (non-repeatable read):

```
    SELECT f2 FROM t1 WHERE f1 = 1;  | SELECT f2 FROM t1 WHERE f1 = 1;
UPDATE t1 SET f2 = f2 + 1 WHERE f1 = 1;
               COMMIT;               |
                                     | SELECT f2 FROM t1 WHERE f1 = 1;
```

- фантомное чтение (phantom read):

```
                                     | SELECT SUM(f2) FROM t1;
INSERT INTO t1 (f1, f2) VALUES (15, 20)
                                     | SELECT SUM(f2) FROM t1;
```

43. Конфликты и эквивалентность расписаний по конфликтам. Сериализуемость

Определение 31. *Конфликт* — пара операций из расписания такая, что

- операции принадлежат разным транзакциям;
- они работают с одним элементом данных;
- по крайней мере одна из двух — операция записи.

Определение 32. Расписания *эквивалентны*, если из множества конфликтов совпадают.

Определение 33. Расписание называется *серийным*, если все операции одной транзакции либо предшествуют, либо следуют за операциями любой другой транзакции.

Определение 34. Расписание называется *сериализуемым* (по конфликтам), если оно эквивалентно серийному.

Определение 35. Считаем, что любые операции чтения коммутируют и любые операции над разными элементами коммутируют. Расписание *сериализуемо* (по коммутативности), если его можно преобразовать в серийное перестановками соседних операций.

44. Использование замков: двухфазный протокол блокирования

Замки для одного элемента данных несовместимы, если они устанавливаются разными транзакциями и по крайней мере один из них на запись. Попытка установить несовместимый замок переводит транзакцию в состояние ожидания.

Протокол блокирования 2PL:

- для каждой операции необходимо предварительно установить замок, все замки должны быть сняты до завершения транзакции;
- транзакция не может устанавливать новые замки после того, как она сняла какой-то из замков.

Двухфазный протокол генерирует расписания, сериализуемые по конфликтам.

45. Тупики: обнаружение и разрешение

Транзакции, попавшие в тупик должны быть оборваны.
Для обнаружения тупиков строится *граф ожиданий*:

- вершины — активные транзакции;
- дуги проводятся из ожидающей транзакции в транзакцию, установившую несовместимые замки.

Тупик имеет место тогда и только тогда, когда в графе ожиданий есть контур.

46. Протокол установки замков для дерева

Пусть БД структурирована как дерево. Тогда можно применить протокол WTL (Write-only tree locking):

- все замки — на запись;
- для установки замка необходимо иметь установленный замок на родительскую вершину;
- ни один элемент не блокируется одной и той же транзакцией дважды;
- снятие замков возможно в любое время и не препятствует установке новых замков.

Утверждения.

1. граф сериализуемости ацикличен;
2. транзакции сериализуются в том порядке, в котором они устанавливают замки на корень;
3. протокол WTL свободен от тупиков.

47. Уровни изоляции транзакций

```
SET TRANSACTION ISOLATION LEVEL
        {READ UNCOMMITED | READ COMMITED | REPEATABLE READ | SERIALIZABLE}
        {READ WRITE | READ ONLY} [NOT] DEFERRABLE;
```

Уровни изоляции:

1. `READ UNCOMMITTED`:
   - не устанавливаются блокировки на чтение;
   - транзакция может считывать изменённые другими транзакциями, но не зафиксированные строки, т. е. можно читать "грязные"данные;
   - значения в данных могут быть изменены и до окончания транзакции строки могут появляться и исчезать в наборе данных.
2. `READ COMMITTED`:
   - нельзя считывать данные, которые были изменены другими транзакциями, но ещё не были зафиксированы;
   - считанные данные могут быть изменены другими транзакциями во время работы текущей транзакции.
3. `REPEATABLE READ`:
   - нельзя считывать незафиксированные данные;
   - совмещаемые блокировки применяются ко всем считываемым данным и сохраняются до завершения; это запрещает другим транзакциям изменять строки, считанные данной транзакцией;
   - другие транзакции могут вставлять новые строки, соответствующие условиям поиска текущей транзакции.
4. `SERIALIZABLE`:
   - нельзя считывать незафиксированные данные;
   - другие транзакции не могут изменять данные, считанные текущей транзакцией;
   - другие транзакции не могут вставлять новые строки, удовлетворяющие условиям поиска текущей транзакции; при повторном считывании будет тот же самый набор строк.

48. Фазы транзакций. Обрывы транзакций

1. установка уровня изоляции;
2. начало

   ```
   BEGIN TRANSACTION
   ```

3. конец

   ```
   COMMIT
   ROLLBACK
   ```

49. Защита от отказов системы: правила ведения журнала

Определение 36. *Журнал транзакций* — это часть БД, в которую поступают данные обо всех изменениях всех объектов БД. Журнал недоступен пользователям СУБД и поддерживается особо тщательно. Форма записи в журнал зависит от СУБД. Обычно там фиксируется:

- номер транзакции;
- состояние транзакции (завершена фиксацией или откатом, не завершена, ожидает);
- точки сохранения;
- команды, составляющие транзакцию.

Правила ведения журнала:

- журнал ведётся последовательно;
- запись происходит до самих действий (write-ahead log);
- регистрируются операции записи, начала и конца транзакции;
- каждая запись содержит данные отката (redo) и "наката" (redo);
- при фиксации запись журнала обязательно "выталкивается" на диск.

50. Ведение журнала: контрольные точки

Savepoint запоминает промежуточную "текущую копию" состояния базы данных для того, чтобы при необходимости можно было вернуться к состоянию БД в точке сохранения: откатить работу от текущего момента до точки сохранения (`ROLLBACK TO sp`) или зафиксировать работу от начала транзакции до точки сохранения (`COMMIT TO sp`). На одну транзакцию может быть несколько точек сохранения.

```
SAVE {TRAN | TRANSACTION} sp_name;
```

51. Технологии клиент—сервер: подходы

Роли клиента и сервера выполняются на разных системах. Различают "толстые" и "тонкие" клиенты. Выделяют четыре подхода к клиент—серверной модели (рис. 1):

- Файловый сервер (File Server):
  - файловый сервер хранит файлы, предоставляя к ним доступ пользователям сети;
  - всё программное обеспечение ИС располагается на сетевых компьютерах;
  - для выполнения операций с данными необходимо получить их копию на сетевой компьютер;
  - недостатки:
    - высокий трафик;
    - узкий спектр операций манипуляций с данными;
    - отсутствие адекватных средств безопасности доступа к данным.
- Доступ к удалённым данным (Remote Access Data):
  - унификация интерфейса "клиент—сервер" в виде языка SQL;
  - перенос компонента представления и прикладного компонента на клиенты существенно разгружает сервер БД;
  - администрирование приложений практически невозможно из-за совмещения в одной программе различных функций.
- Сервер базы данных (DataBase Server):
  - основа — механизм хранимых процедур. Процедуры хранятся в словаре базы данных, разделяются между несколькими клиентами и выполняются на сервере;
  - достоинства:
    - возможность централизованного администрирования прикладных функций;

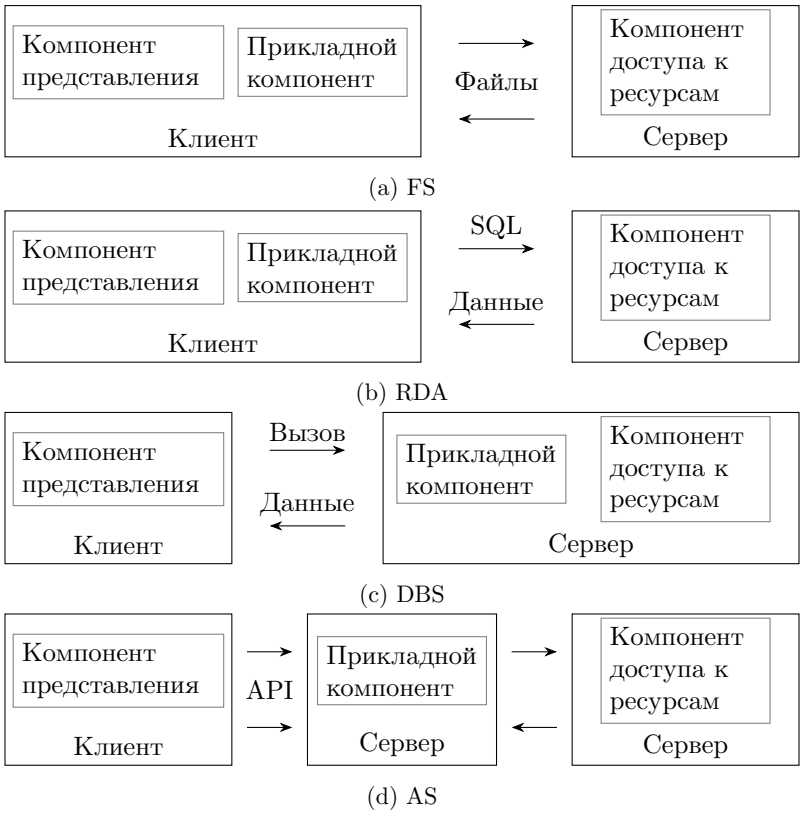

Рис. 1: Подходы к клиент—серверной модели.

    - снижение трафика (вместо SQL-запросов по сети направляются вызовы процедур);
    - возможность разделения процедуры между несколькими приложениями;
    - экономия ресурсов сервера за счёт использования единожды созданного плана выполнения процедуры.
  - недостатки:
    - ограниченность средств, используемых для написания процедур;
    - сфера их использования ограничена конкретной СУБД;
    - не во всех СУБД есть возможность отладки и тестирования хранимых процедур.
  - обычно используется модель RDA+DSB;
- Сервер приложений (Application Server):
  - специфические функции приложений выделятся на отдельный сервер — *сервер приложений*;
  - достоинства:
    - существует большое количество компонентов;
    - массовое использование в относительно простых системах;
    - средства генерации кода.
  - недостатки:
    - неэффективное использование серверов данных;
    - слишком большое количество сетевых обменов;
    - искусственное привязывание ролей к слоям.