Основы проектирования баз данных

Работа с данными и структурой БД

Поломошнов И.С.

преп. каф. ИБ, ИМФКН, БГУ

1. Создание таблиц в СУБД (DDL — Data Definition Language)

DDL используется для определения структуры базы данных. Основные команды: CREATE, ALTER, DROP.

Синтаксис CREATE TABLE

```
CREATE TABLE table_name (
    column1 datatype constraints,
    column2 datatype constraints,
    ...
    table_constraints
);
```

Пример создания таблицы "Сотрудники"

```
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE,
    hire_date DATE DEFAULT CURRENT_DATE,
    salary DECIMAL(10, 2) CHECK (salary > 0),
    department_id INTEGER REFERENCES departments(id)
);
```

Изменение таблицы (ALTER)

```sql
-- Добавление столбца
ALTER TABLE employees ADD COLUMN phone VARCHAR(20);

-- Изменение типа данных столбца
ALTER TABLE employees ALTER COLUMN salary TYPE NUMERIC(12, 2);

-- Удаление столбца
ALTER TABLE employees DROP COLUMN phone;
```

Удаление таблицы (DROP)

```
-- Полное удаление таблицы
DROP TABLE employees;

-- Удаление только данных (очистка)
TRUNCATE TABLE employees;
```

2. Заполнение таблиц (DML — Data Manipulation Language)

DML используется для работы с данными внутри таблиц.

Вставка данных (INSERT)

```sql
-- Вставка одной строки
INSERT INTO employees (first_name, last_name, email, salary, department_id)
VALUES ('Иван', 'Петров', 'ivan@company.com', 75000, 1);

-- Вставка нескольких строк
INSERT INTO employees (first_name, last_name, email, salary) VALUES
    ('Мария', 'Сидорова', 'maria@company.com', 82000),
    ('Петр', 'Иванов', 'petr@company.com', 69000);

-- Вставка с использованием подзапроса
INSERT INTO employees_archive (id, first_name, last_name, email)
SELECT id, first_name, last_name, email FROM employees WHERE hire_date < '2020-01-01';
```

Обновление данных (UPDATE)

```sql
-- Обновление одного поля
UPDATE employees
SET salary = 80000
WHERE id = 1;

-- Обновление нескольких полей
UPDATE employees
SET salary = salary * 1.1,
    last_updated = NOW()
WHERE department_id = 2;
```

Удаление данных (DELETE)

```sql
-- Удаление конкретной записи
DELETE FROM employees WHERE id = 3;
```

```sql
-- Удаление всех записей, удовлетворяющих условию
DELETE FROM employees WHERE department_id IS NULL;
```

3. Простые запросы SELECT. Сортировка и фильтрация

Базовая выборка (SELECT)

```sql
-- Выбрать все столбцы
SELECT * FROM employees;

-- Выбрать конкретные столбцы
SELECT first_name, last_name, email FROM employees;

-- Выбрать с переименованием столбцов (алиасы)
SELECT
    first_name AS "Имя",
    last_name AS "Фамилия",
    salary AS "Зарплата"
FROM employees;
```

Фильтрация (WHERE)

```sql
-- Числовые сравнения
SELECT * FROM employees WHERE salary > 50000;

-- Текстовый поиск
SELECT * FROM employees WHERE last_name = 'Петров';

-- Логические операторы
SELECT * FROM employees
WHERE salary > 60000 AND department_id = 1;

-- Проверка на NULL
SELECT * FROM employees WHERE department_id IS NULL;

-- Оператор IN
SELECT * FROM employees
WHERE department_id IN (1, 3, 5);

-- Поиск по шаблону (LIKE)
SELECT * FROM employees
WHERE email LIKE '%@company.com';
```

Сортировка (ORDER BY)

```sql
-- Сортировка по возрастанию (по умолчанию)
SELECT * FROM employees ORDER BY last_name;

-- Сортировка по убыванию
SELECT * FROM employees ORDER BY salary DESC;

-- Сортировка по нескольким полям
SELECT * FROM employees
ORDER BY department_id ASC, salary DESC;

-- Ограничение количества записей
SELECT * FROM employees
ORDER BY hire_date DESC
LIMIT 10;
```

4. Группировка данных и агрегатные функции

Агрегатные функции

```sql
-- Подсчет количества записей
SELECT COUNT(*) FROM employees;

-- Сумма значений
SELECT SUM(salary) AS total_salary FROM employees;

-- Среднее значение
SELECT AVG(salary) AS avg_salary FROM employees;

-- Минимальное и максимальное значение
SELECT
    MIN(salary) AS min_salary,
    MAX(salary) AS max_salary
FROM employees;
```

Группировка (GROUP BY)

```sql
-- Подсчет сотрудников по отделам
SELECT
    department_id,
    COUNT(*) AS employee_count,
    AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id;

-- Группировка с фильтрацией групп (HAVING)
SELECT
    department_id,
    COUNT(*) AS employee_count,
    AVG(salary) AS avg_salary
FROM employees
GROUP BY department_id
HAVING AVG(salary) > 70000;
```

5. Соединение таблиц (JOIN)

Виды JOIN

- **INNER JOIN** — только совпадающие записи
- **LEFT JOIN** — все записи из левой таблицы + совпадающие из правой
- **RIGHT JOIN** — все записи из правой таблицы + совпадающие из левой
- **FULL OUTER JOIN** — все записи из обеих таблиц

Примеры JOIN

```sql
-- INNER JOIN: сотрудники с названиями отделов
SELECT
    e.first_name,
    e.last_name,
    d.name AS department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.id;
```

```sql
-- LEFT JOIN: все сотрудники, включая тех, у кого нет отдела
SELECT
    e.first_name,
    e.last_name,
    d.name AS department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.id;
```

```sql
-- JOIN нескольких таблиц
SELECT
    e.first_name,
    e.last_name,
    d.name AS department,
    p.name AS project_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.id
INNER JOIN employee_projects ep ON e.id = ep.employee_id
INNER JOIN projects p ON ep.project_id = p.id;
```

6. Создание представлений (VIEW) и индексов

Представления (VIEW)

Виртуальные таблицы, основанные на результате SELECT-запроса.

```sql
-- Создание простого представления
CREATE VIEW v_employee_details AS
SELECT
    e.id,
    e.first_name || ' ' || e.last_name AS full_name,
    e.email,
    d.name AS department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.id;

-- Использование представления
SELECT * FROM v_employee_details WHERE department_name = 'IT';
```

```sql
-- Создание материализованного представления (PostgreSQL)
CREATE MATERIALIZED VIEW mv_department_stats AS
SELECT
    d.id,
    d.name,
    COUNT(e.id) AS employee_count,
    AVG(e.salary) AS avg_salary
FROM departments d
LEFT JOIN employees e ON d.id = e.department_id
GROUP BY d.id, d.name;

-- Обновление материализованного представления
REFRESH MATERIALIZED VIEW mv_department_stats;

-- Удаление представления
DROP VIEW v_employee_details;
```

Индексы

Ускоряют поиск данных, но замедляют вставку/обновление.

```sql
-- Создание индекса для одного столбца
CREATE INDEX idx_employees_last_name
ON employees(last_name);

-- Уникальный индекс
CREATE UNIQUE INDEX idx_employees_email
ON employees(email);

-- Составной индекс (для запросов с несколькими условиями)
CREATE INDEX idx_employees_dept_salary
ON employees(department_id, salary);

-- Частичный индекс (только для части данных)
CREATE INDEX idx_employees_high_salary
ON employees(salary)
WHERE salary > 100000;

-- Проверка использования индекса (EXPLAIN)
EXPLAIN SELECT * FROM employees WHERE last_name = 'Петров';

-- Удаление индекса
DROP INDEX idx_employees_last_name;
```

Полезные советы

1. Всегда используйте явные имена столбцов вместо SELECT * в продакшене
2. Для сложных запросов используйте EXPLAIN для анализа производительности
3. Индексы эффективны для столбцов, часто используемых в WHERE и JOIN
4. Нормализуйте данные, но не бойтесь денормализации для сложных отчетов
5. Регулярно делайте бэкапы перед выполнением DROP или массовых UPDATE/DELETE

Спасибо за внимание

Вопросы?