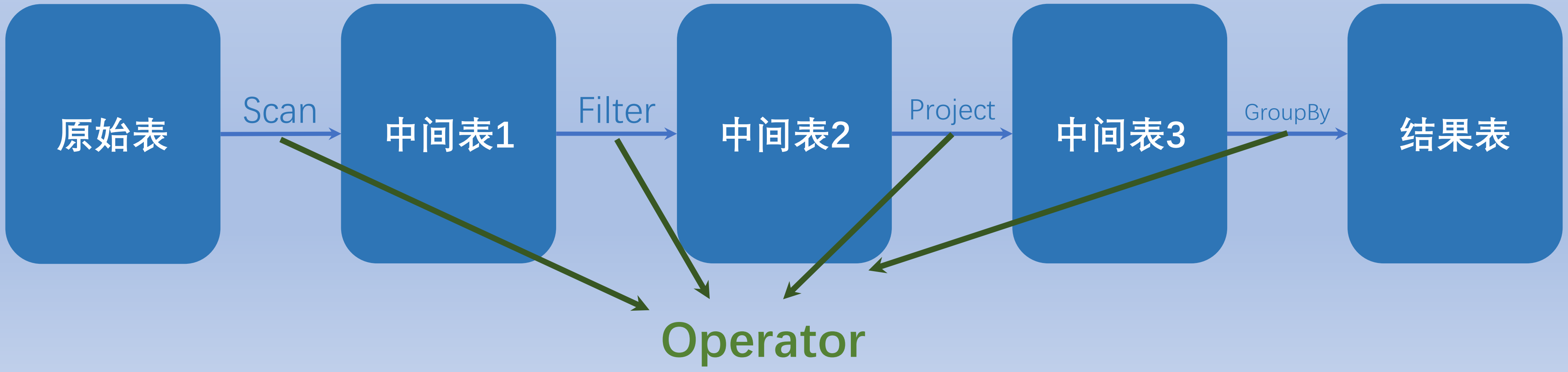
面向对象的数据库设计

- 封装、继承、多态：从入门到~~放弃~~精通
- 数据库操作 select

```
select ps_partkey, ps_availqty
from partsupp
where ps_availqty < 3000 and ps_suppkey < 1000
group by partkey
```



封装 Encapsulation

```
class Operator {
protected:
    RowTable *tab_in[4] = { NULL, NULL, NULL, NULL };
    RowTable *tab_out   = NULL;
public:
    virtual ~Operator() = default;
    virtual bool init() = 0;
    virtual bool getNext(char *ptr) = 0;
    virtual bool isEnd() = 0;
    virtual bool close() = 0;
    RowTable *getRowTableOut();
}
```

隐藏：保护私有数据

别人并不需要知道我的输入表是什么

Member Function (C++)

Method (Java/Python)

子类 Operator 需要重写（Override）

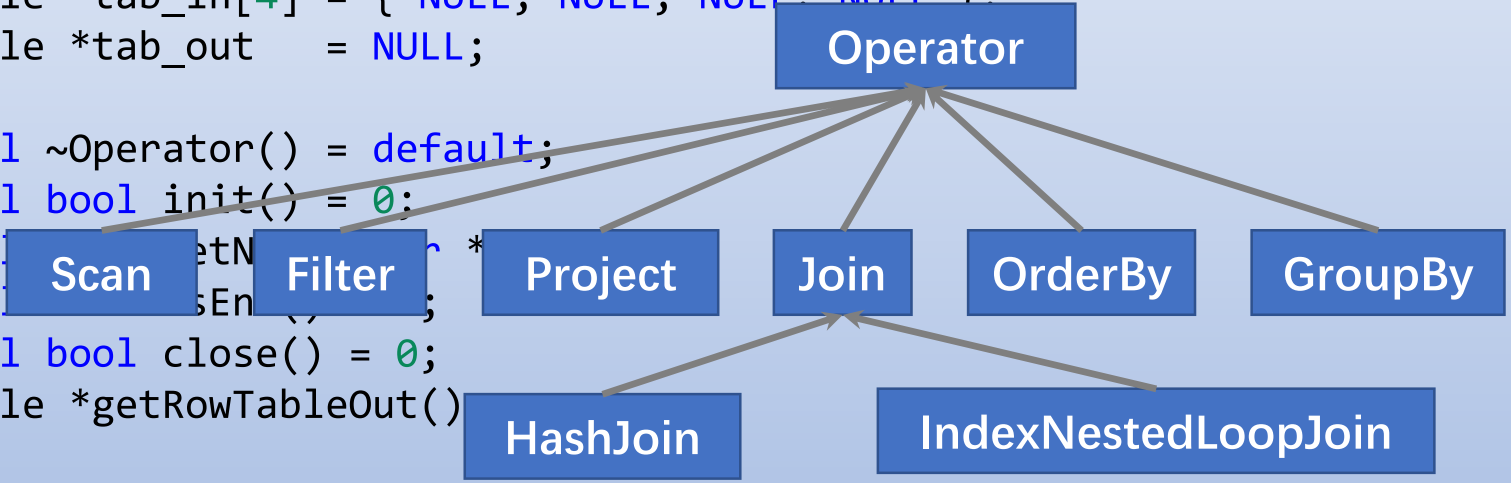
Java

```
public class Apple extends Fruit {
    @Override
    public void show_name(){
        System.out.println("Apple");
    }
}
```

继承！

继承 Inheritance

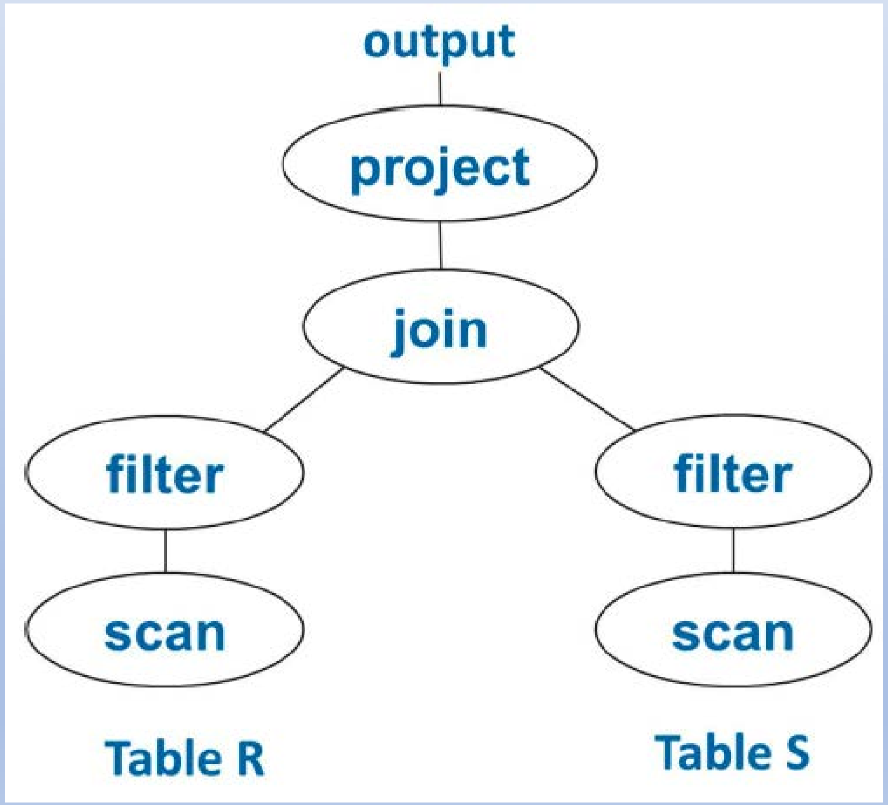
```
class Operator {
protected:
    RowTable *tab_in[4] = { NULL, NULL, NULL, NULL };
    RowTable *tab_out   = NULL;
public:
    virtual ~Operator() = default;
    virtual bool init() = 0;
    virtual       etN        r *
    virtual       sEn  ();
    virtual bool close() = 0;
    RowTable *getRowTableOut()
}
```



- 对于所有 Operator
  - 实现 getRowTableOut() --> 暴露出输出格式
  - 实现其余方法 --> 输出统一、完整的数据

更进一步

- Operator Tree



来源：陈世敏老师的数据库系统 PPT

- `Project(Operator*, int, RequestColumn*);`
- `Join(int, Operator**, int, Condition**);`
- `Filter(Operator*, Condition*);`
- `Filter(Operator*, Condition*, int, RequestColumn*);`

- 输入类型为 `Operator*`
- 该调用哪个子类的 `getNext()` 呢？

程序员的伟大发明：多态！

多态 Polymorphism

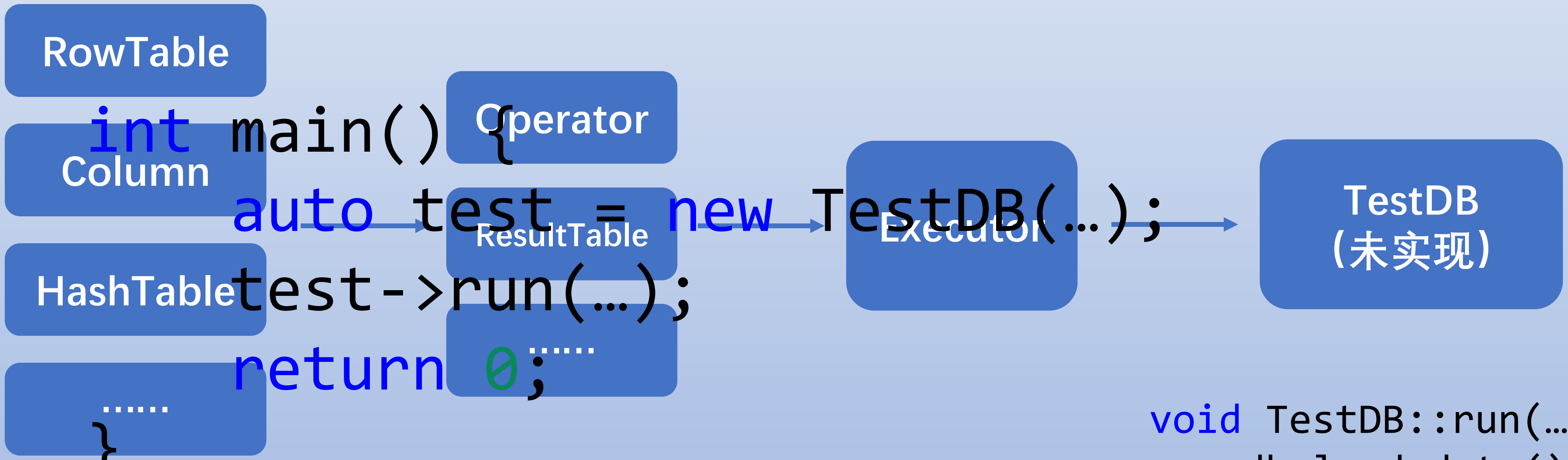
- 编译时多态性（静态多态）
  - 通过重载（Overload）函数实现
    - `Boolean foo(int bar)`
    - `Boolean foo(String name)`
- 运行时多态性（动态多态）
  - C++
    - 继承、重写、指针或引用
    - 通过**基类指针或引用**调用虚函数时，会调用**当前对象的实际类型**中声明的函数
  - Java
    - 继承、重写、父类引用指向子类对象
- 不管是什么 Operator 的指针，只需要 `op->getNext()`

如何用简单部件构造复杂系统

```
int Executor::exec(SelectQuery *query, ResultTable *result) {
    // 构建 Operator Tree
    Operator *top[4];
    for (int i = 0; i < select_num; i++)
        top[i] = new Scan(g_catalog.getObjByName(name[i]));
    top[0] = new Join(tab_number, top, count, cond);
    // ……
    result_op = top[0];
    // 从结果 Operator 中读出数据并返回结果数量
    // ……
    return num;
}
```

结局

- 使用小模块搭建成大系统



```
int main() {
    auto test = new TestDB(…);
    test->run(…);
    return 0;
}
```

```
void TestDB::run(…) {
    db.load_data();
    Executor executor;
    executor.exec(query);
    check_result();
}
```