

# *VT Sistem Gerçeklemesi Ders*

## *Notları- #9*

**Remote:** Kullanıcıdan gelen JDBC isteklerini karşılar.

**Planner:** SQL ifadesi için işleme planı oluşturur ve karşılık gelen ilişkisel cebir ifadesini oluşturur.

**Parse:** SQL ifadesindeki tablo, nitelik ve ifadeleri ayrıştırır.

**Query:** Algebra ile ifade edilen sorguları gerçekler.

**Metadata:** Tablolara ait katalog bilgilerini organize eder.

**Record:** disk sayfalarına yazma/okumayı kayıt seviyesinde gerçekler.

**Transaction&Recovery:** Eşzamanlılık için gerekli olan disk sayfa erişimi kısıtlamalarını organize eder ve veri kurtarma için kayıt\_defteri (*log*) dosyalarına bilgi girer.

**Buffer:** En sık/son erişilen disk sayfalarını ana hafıza tampon bölgede tutmak için gerekli işlemleri yapar.

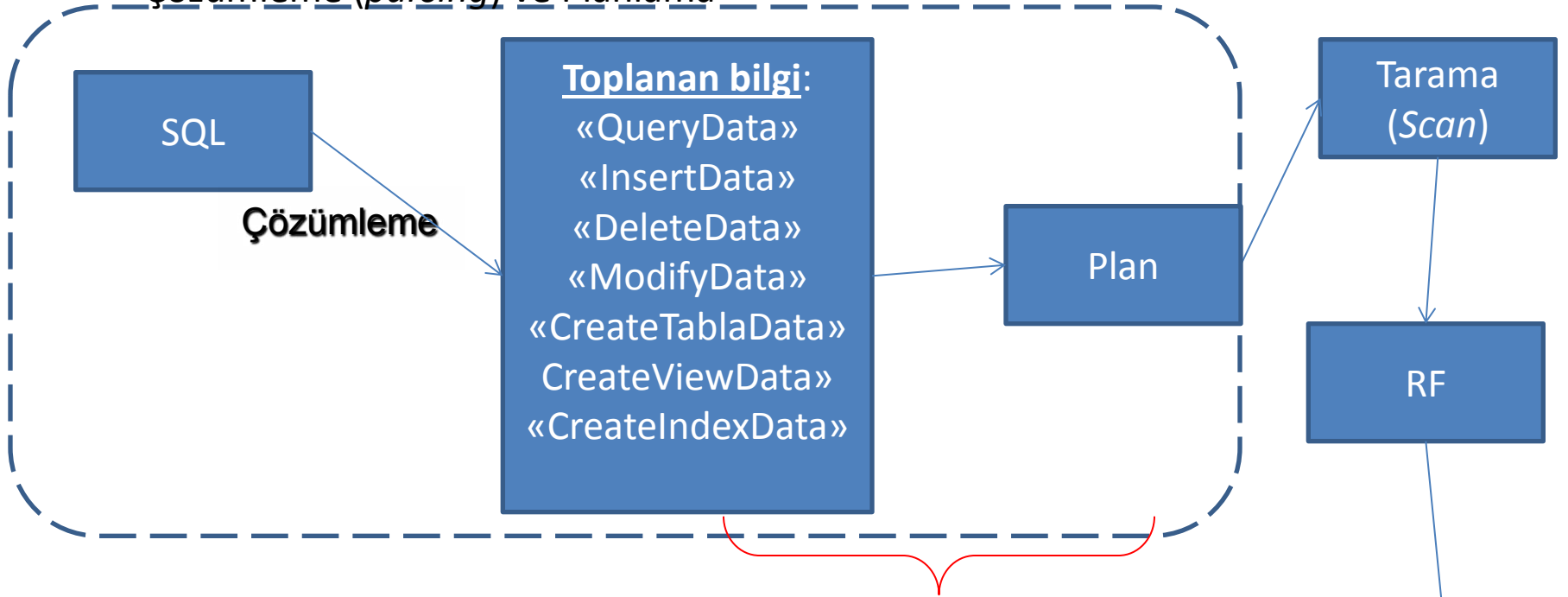
**Log:** Kayıt\_defterine bilgi yazılmasını ve taranması işlemlerini düzenler.

**File:** Dosya blokları ile ana hafıza sayfaları arasında bilgi transferini organize eder.

- Planlama ne yapar?
- SimpleDB'de Sorgu Planlaması
- Standard VTYS'de sorgu planlaması
- SimpleDB'de Yenilemelerin Planlanması
- SimpleDB Planner Gerçeklemesi

# Neredeyiz?

## Çözümleme (*parsing*) ve Planlama



1. SQL cümlesi doğrulama
2. (SimpleDB'de) karşılık gelen Plan oluşturma  
(Standart bir VTYS'de) Planlar oluşturup en iyi planın seçilmesi

# SimpleDB'de Planner API ve bir Örnek:

```
public Plan createQueryPlan(String query, Transaction tx);  
public int  executeUpdate(String cmd, Transaction tx);
```

```
SimpleDB.init("studentdb");  
Planner planner = SimpleDB.planner();  
Transaction tx  = new Transaction();  
  
// part 1: Process a query  
String qry = "select sname, gradyear from student";  
Plan p = planner.createQueryPlan(qry, tx);  
Scan s = p.open();  
while (s.next())  
    System.out.println(s.getString("sname") + " " +  
                       s.getInt("gradyear"));  
s.close();  
  
// part 2: Process an update command  
String cmd = "delete from STUDENT where MajorId = 30";  
int num = planner.executeUpdate(cmd, tx);  
System.out.println(num + " students were deleted");  
  
tx.commit();
```

**Figure 19-2**

Using the SimpleDB planner

# Planner ne yapar?

1. *Parse the SQL statement.* Call the parser, passing it the input string; the parser will return an object containing the data from the SQL statement. For example, the parser will return a *QueryData* object for a query, an *InsertData* object for an insert statement, and so on.
2. *Verify the SQL statement.* Examine the *QueryData* (or *InsertData*, etc.) object to determine if it is semantically meaningful.
3. *Create a plan for the SQL statement.* Use a planning algorithm to determine a query tree corresponding to the statement and create a plan corresponding to that tree.
- 4a. *Return the plan* (for the *createQueryPlan* method).
- 4b. *Execute the plan* (for the *executeUpdate* method). Create a scan by opening the plan; then iterate through the scan, making the appropriate update for each record in the scan and returning the number of records affected.

## **Figure 19-3**

The steps taken by the two planner methods

# SimpleDB'de Sorgu Planlama

## SIMPLEDB PLAN ALGORİTMASI:

1.) *from* cümleciğindeki her bir T tablosuna karşılık gelen PLAN oluşturma

- T bir saklı bir tablo ise, plan TABLEPLAN olur.
- T bir görüntü ise bu algoritma T'nin görüntüsü için uygulanır.

2.) 1. adımda oluşan PLAN'ların PRODUCT edilmesi

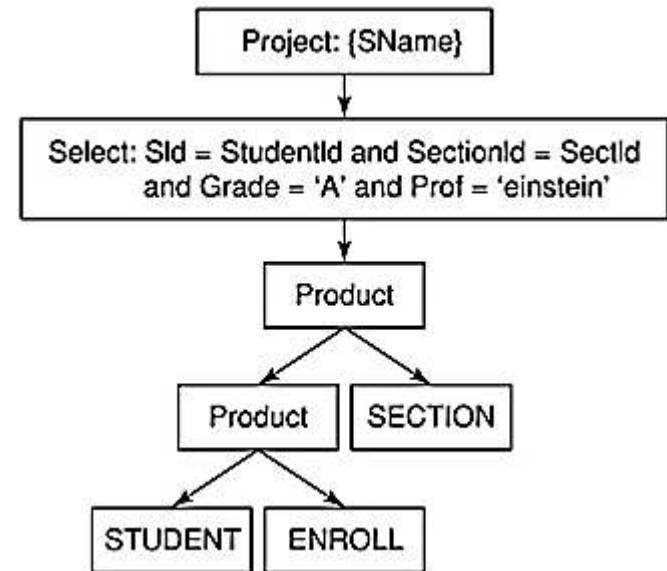
3.) *where* cümleciğindeki yüklemnin uygulanması

4.) *select* cümleciğindeki nitelikler için PROJECT operasyonu

- SimpleDB SQL'de:
  - Select
  - From
  - where
- SimpleDB SQL'de olmayanlar:
  - Toplamı ilgilendiren operasyonlar (Aggregate f.)
  - Sıralama (order by)
  - Gruplama (group by)
  - İç içe sorgulama (nested Q)
  - İsimlendirme (AS ..)

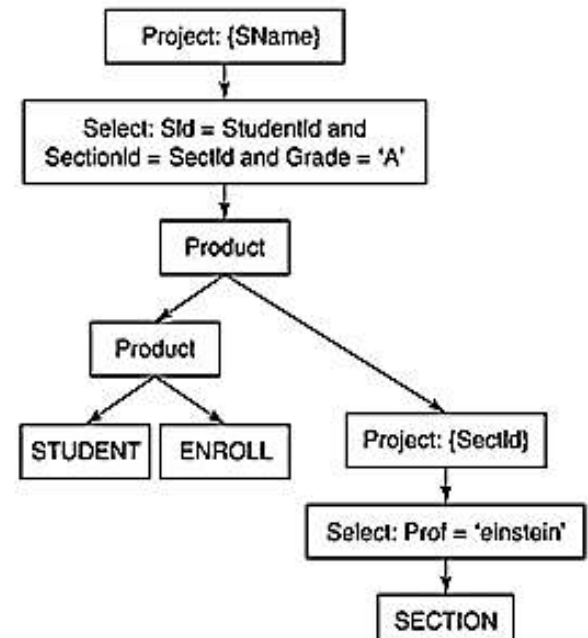
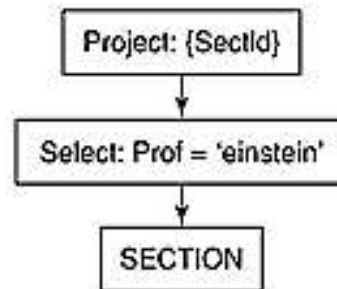
## 2 Örnek

```
select SName
from STUDENT, ENROLL, SECTION
where SId=StudentId
and SectionId=SectId
and Grade='A'
and Prof='einstein'
```



```
create view EINSTEIN as
select SectId
from SECTION
where Prof='einstein'
```

```
select SName
from STUDENT, ENROLL, EINSTEIN
where SId=StudentId
and SectionId=SectId
and Grade='A'
```



# SimpleDB'de Sorgu Planlama Gerçekleme

```
public class BasicQueryPlanner implements QueryPlanner

    public Plan createPlan(QueryData data, Transaction tx)

//Step 1: Create a plan for each mentioned table or vi
    List<Plan> plans = new ArrayList<Plan>();
    for (String tblname : data.tables()) {
        String viewdef =
            SimpleDB.mdMgr().getViewDef(tblname, tx);
        if (viewdef != null)
            plans.add(SimpleDB.planner()
                .createQueryPlan(viewdef, tx));
        else
            plans.add(new TablePlan(tblname, tx));
    }

//Step 2: Create the product of all plans
    Plan p = plans.remove(0);
    for (Plan nextplan : plans)
        p = new ProductPlan(p, nextplan);

//Step 3: Add a select plan for the predicate
    p = new SelectPlan(p, data.pred());

//Step 4: Project on the field names
    p = new ProjectPlan(p, data.fields());
    return p;
}
```

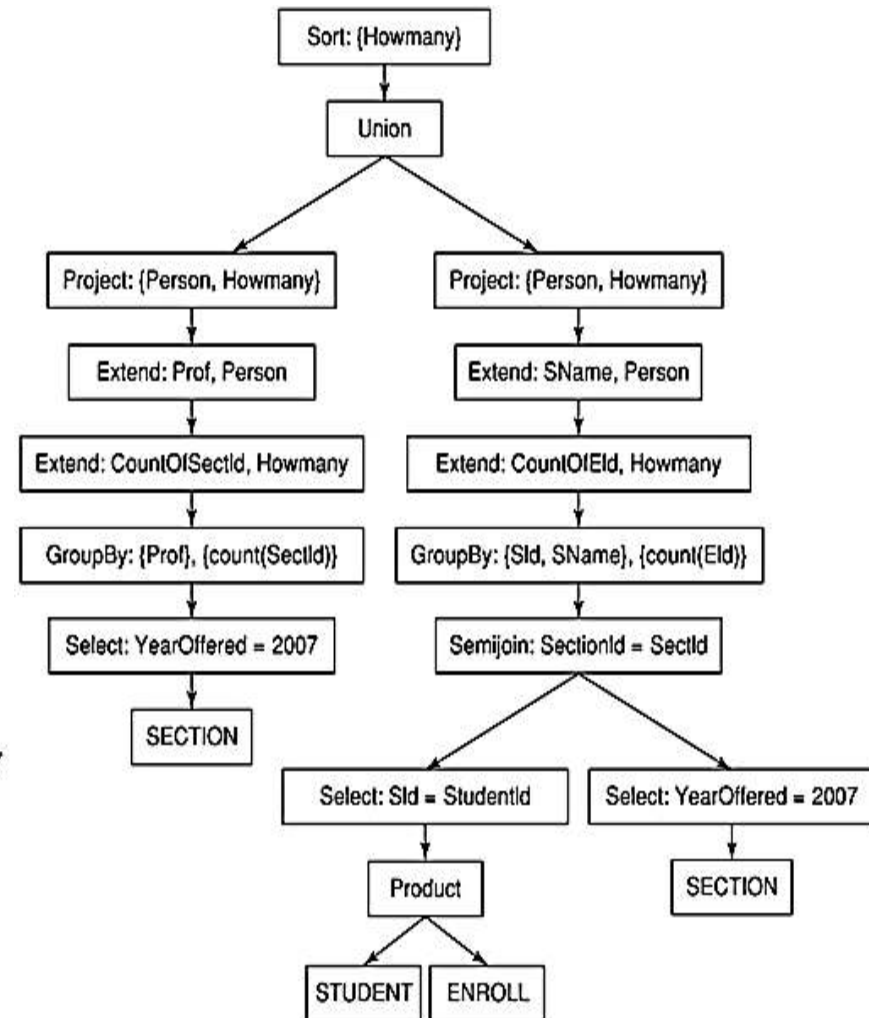
# Standard SQL Sorgu Planlama

- Product/join → outerjoin → select, semijoin, antijoin  
→ group by → extend → project → union → sort

```
select SName as Person, count(EId) as Howmany
from STUDENT, ENROLL
where SId=StudentId
and SectionId in
  (select SectId
   from SECTION
   where YearOffered=2007)
group by SId, SName
```

union

```
select Prof as Person, count(SectId) as Howmany
from SECTION
where YearOffered=2007
group by Prof
order by Howmany
```





# SimpleDB'ye eklentiler (*Parser, Planner, Scan/Plan*)

- Önceki sunudaki SQL sorgusunun çalıştırılması için; SimpleDB dilbilgisi kurallarına yeni eklentiler ve mevcutta bazı değişiklikler yapılmalıdır.
- Örneğin: «Query» kategorisine, «order by» ve «group by» kategorileri dahil edilmelidir.
- Bununla beraber Çözümleyici'deki (*Parser*) yapılan her değişiklik ve eklenti, Planner tarafında yeni eklentiler gerektirir.
- Ağaçtaki her bir düğüme karşılık gelen bir Scan/Plan gerçekleştirilmesi gerektirir.

# SimpleDB'de Yenilemelerin Planlanması

```
public class BasicUpdatePlanner implements UpdatePlanner {

    public int executeDelete(DeleteData data,
                             Transaction tx) {
        Plan p = new TablePlan(data.tableName(), tx);
        p = new SelectPlan(p, data.pred());
        UpdateScan us = (UpdateScan) p.open();
        int count = 0;
        while(us.next()) {
            us.delete();
            count++;
        }
        us.close();
        return count;
    }

    public int executeModify(ModifyData data,
                             Transaction tx) {
        Plan p = new TablePlan(data.tableName(), tx);
        p = new SelectPlan(p, data.pred());
        UpdateScan us = (UpdateScan) p.open();
        int count = 0;
        while(us.next()) {
            Constant val = data.newValue().evaluate(us);
            us.setVal(data.targetField(), val);
            count++;
        }
        us.close();
        return count;
    }

    public int executeInsert(InsertData data,
                             Transaction tx) {
        Plan p = new TablePlan(data.tableName(), tx);
        UpdateScan us = (UpdateScan) p.open();
```

**Figure 19-9**

The code for the SimpleDB class *BasicUpdatePlanner*

```
        us.insert();
        Iterator<Constant> iter = data.vals().iterator();
        for (String fldname : data.fields()) {
            Constant val = iter.next();
            us.setVal(fldname, val);
        }
        us.close();
        return 1;
    }

    public int executeCreateTable(CreateTableData data,
                                   Transaction tx){
        SimpleDB.mdMgr().createTable(data.tableName(),
                                      data.newSchema(), tx);

        return 0;
    }

    public int executeCreateView(CreateViewData data,
                                   Transaction tx) {
        SimpleDB.mdMgr().createView(data.viewName(),
                                      data.viewDef(), tx);

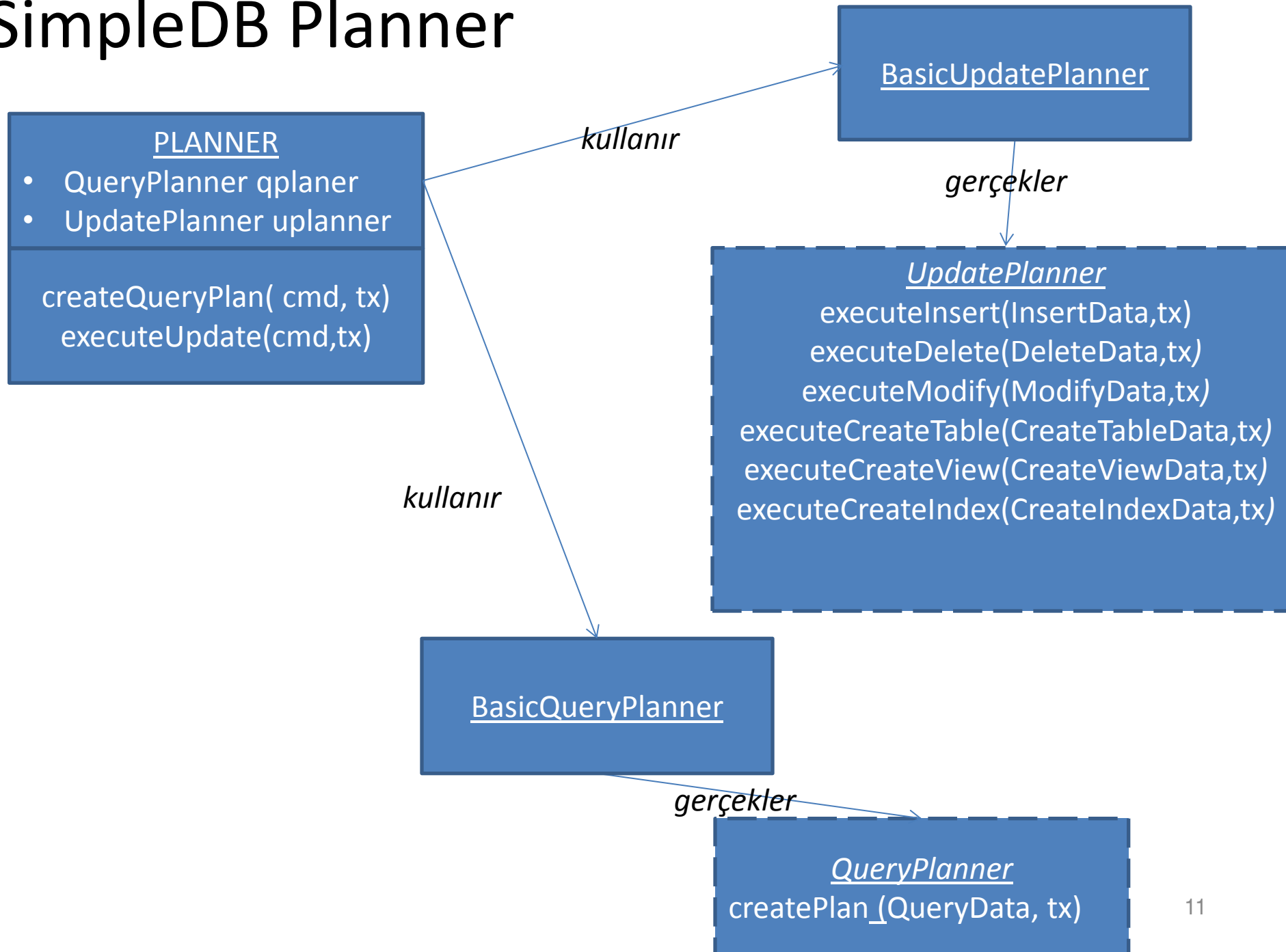
        return 0;
    }

    public int executeCreateIndex(CreateIndexData data,
                                   Transaction tx){
        SimpleDB.mdMgr().createIndex(data.indexName(),
                                      data.tableName(), data.fieldName(), tx);

        return 0;
    }
}
```

**Figure 19-9** (Continued)

# SimpleDB Planner



# SimpleDB Sistem Başlatılmasında Planner başlatılması

```
public static Planner planner() {
    QueryPlanner qplanner = new BasicQueryPlanner();
    UpdatePlanner uplanner = new BasicUpdatePlanner();
    return new Planner(qplanner, uplanner);
}

public class Planner {
    private QueryPlanner qplanner;
    private UpdatePlanner uplanner;

    public Planner(QueryPlanner qplanner,
                  UpdatePlanner uplanner) {
        this.qplanner = qplanner;
        this.uplanner = uplanner;
    }

    public Plan createQueryPlan(String cmd, Transaction tx) {
        Parser parser = new Parser(cmd);
        QueryData data = parser.query();
        // code to verify the query should be here...
        return qplanner.createPlan(data, tx);
    }

    public int executeUpdate(String cmd, Transaction tx) {
        Parser parser = new Parser(cmd);
        Object obj = parser.updateCmd();
        // code to verify the update should be here ...

        if (obj instanceof InsertData)
            return uplanner.executeInsert(
                (InsertData)obj, tx);
        else if (obj instanceof DeleteData)
            return uplanner.executeDelete(
                (DeleteData)obj, tx);
        else if (obj instanceof ModifyData)
            return uplanner.executeModify(
                (ModifyData)obj, tx);
        else if (obj instanceof CreateTableData)
            return uplanner.executeCreateTable(
                (CreateTableData)obj, tx);
        else if (obj instanceof CreateViewData)
            return uplanner.executeCreateView(
                (CreateViewData)obj, tx);
        else if (obj instanceof CreateIndexData)
            return uplanner.executeCreateIndex(
                (CreateIndexData)obj, tx);
        else
            return 0; // this option should never occur
    }
}
```