VT Sistem Gerçeklemesi Ders Notları- #8

Remote: Kullanıcıdan gelen JDBC isteklerini karşılar.

Planner: SQL ifadesi için işleme planı oluşturur ve karşılık gelen ilşkisel 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.

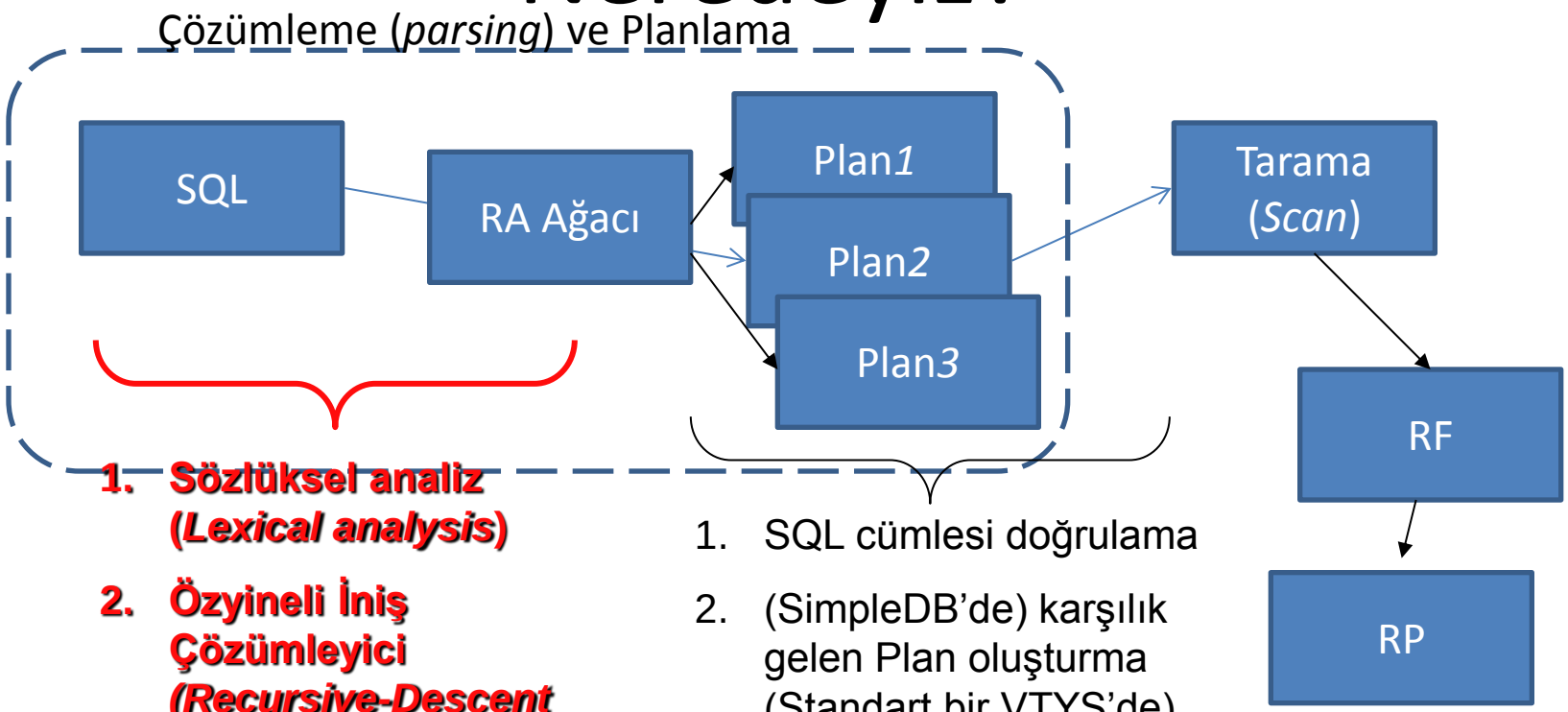
Log: Kayıt_defterine bilgi yazılmasını ve taranması işlemlerini düzenler.

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

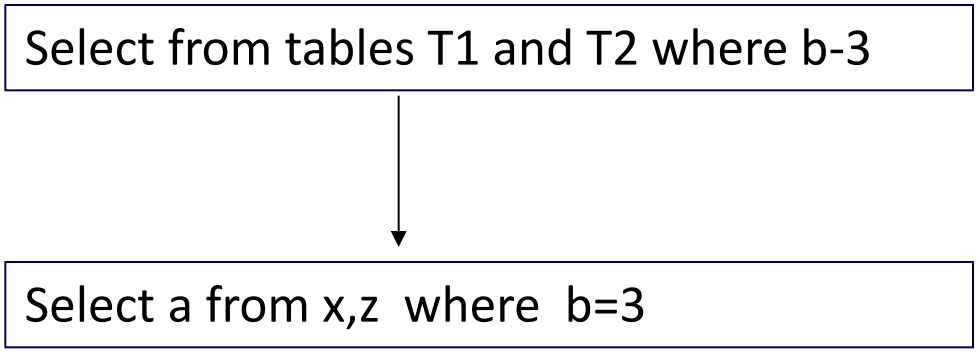
Sorgu Çözümleme

- Sözdizimi (*syntax*) ve anlamsal (*semantic*)
- Sözlüksel analiz (*Lexical analysis*)
- Dilbilgisi (*Grammer*)
- Özyineli İniş Çözümleyici *(Recursive-Decent Parsing)*
- *SQL cümlelerinin çözümlenmesi*
  - *Sorgu, Yenileme, Ekleme, Silme, Tablo/Görüntü/Index Oluşturma cümlelerinin çözümlenmesi*

Neredeyiz?


Çözümleme (*parsing*) ve Planlama
SQL
RA Ağacı
Plan*1*
Plan*2*
Plan*3*
Tarama
(*Scan*)
RF
RP
1. **Sözlüksel analiz (*Lexical analysis*)**
2. **Özyineli İniş Çözümleyici *(Recursive-Descent Parsing)***
1. SQL cümlesi doğrulama
2. (SimpleDB'de) karşılık gelen Plan oluşturma (Standart bir VTYS'de) Planlar oluşturup en iyi planın seçilmesi

Sözdizimi (*syntax*) ve anlamsal (*semantic*)



- Anlamsal analiz, sözdizimi kurallarına uygun ifadenin **anlamını** analiz eder, açığa çıkartır.
- Yukarıdaki ilk ifade, sözdizim kurallarına uygun değil. Sözdizim hataları var. Bunu kontrol eden birim: **Çözümleyici** *(sözlüksel ve sözdizimi analizi)*
- İkinci ifade, çözümlemeden geçer; fakat anlamsal analizi için VT üstveri bilgilerini ihtiyaç duyulur. Tablonun var olup/olmaması, tip uyumluluğu gibi.. Bu analizleri yapan ünite ise: **Planlama'dır.** *(planlamanin ilk aşaması olan doğrulamadır)*

Sözdizim (Çözümleme)-1: Sözlüksel analiz *(lexical anaysis)*

- sözlüksel öğe (*token*)
  - Tip
  - Değer
- 5 tip öğe:
  - Tek-karakterli sınırlayıcı: delimiter *(ör: , =)*
  - Tam sayı sabitleri: integer constants *(ör: 123)*
  - Dizgi sabitleri: string constants *(ör: 'yildiz')*
  - Anahtar kelimeler: keywords *(ör: select, from, where)*
  - Belirleyici, değişken ismi: identifiers *(ör: x, STUDENT)*
- Örnek:

Select a from x,z where b=3

TYPE	VALUE
keyword	select
identifier	a
keyword	from
identifier	x
delimiter	,
identifier	z
keyword	where
identifier	b
delimiter	=
intconstant	3

Sözlüksel analiz gerçekleme

```
public boolean matchDelim(char d);
public boolean matchIntConstant();
public boolean matchStringConstant();
public boolean matchKeyword(String w);
public boolean matchId();

public void   eatDelim(char d);
public int    eatIntConstant();
public String eatStringConstant();
public void   eatKeyword(String w);
public String eatId();
```

Figure 18-1
The API for the SimpleDB class *Lexer*

```
public class Lexer {
   private Collection<String> keywords;
   private StreamTokenizer tok;

   public Lexer(String s) {
      initKeywords();
      tok = new StreamTokenizer(new StringReader(s));
      tok.ordinaryChar('.');
      tok.lowerCaseMode(true); nextToken();
   }

//Methods to check the status of the current token

   public boolean matchDelim(char d) {
      return d == (char)tok.ttype;
   }

   public boolean matchIntConstant() {
```

Sözlüksel analiz gerçekleme

```
      return tok.ttype == StreamTokenizer.TT_NUMBER;
   }

   public boolean matchStringConstant() {
      return '\'' == (char)tok.ttype;
   }

   public boolean matchKeyword(String w) {
      return tok.ttype == StreamTokenizer.TT_WORD &&
             tok.sval.equals(w);
   }

   public boolean matchId() {
      return tok.ttype == StreamTokenizer.TT_WORD &&
             !keywords.contains(tok.sval);
   }

//Methods to "eat" the current token

   public void eatDelim(char d) {
      if (!matchDelim(d))
         throw new BadSyntaxException();
      nextToken();
   }

   public int eatIntConstant() {
      if (!matchIntConstant())
         throw new BadSyntaxException();
      int i = (int) tok.nval;
      nextToken();
      return i;
   }

   public String eatStringConstant() {
      if (!matchStringConstant())
         throw new BadSyntaxException();
      String s = tok.sval;
      nextToken();
      return s;
   }

   public void eatKeyword(String w) {
      if (!matchKeyword(w))
         throw new BadSyntaxException();
      nextToken();
   }
```

Figure 18-2 *(Continued)*

```
   public String eatId() {
      if (!matchId())
         throw new BadSyntaxException();
      String s = tok.sval;
      nextToken();
      return s;
   }

   private void nextToken() {
      try {
         tok.nextToken();
      }
      catch(IOException e) {
         throw new BadSyntaxException();
      }
   }

   private void initKeywords() {
      keywords = Arrays.asList("select", "from", "where",
                "and", "insert", "into", "values", "int",
                "varchar", "update", "set", "delete", "on",
                "create", "table", "view", "as", "index");
   }
}
```

Figure 18-2 *(Continued)*

Sözdizim (Çözümleme)-2) Özyineli İniş Çözümleme

- DİLBİLGİSİ (*GRAMMER*): Öğelerin (*tokens*) birbiri arkası dizilmelerindeki kurallar, SÖZDİZİM KATEGORİLERİ (***syntactic cetagories***) ile belirlenir.
- **Kural** (**Rule**):
  - Sol Taraf: <Kategori_İsmi >
  - Sağ Taraf:
    - *öğeler(tokens)*
    - diğer kategoriler
    - Bazı özel karakterler: |, [,],(,)
- **Sağlanmayanlar:**
  - Tip uyumluluğu
  - Liste eleman sayılarının uyumluluğ

```
<Field>       := IdTok
<Constant>    := StrTok | IntTok
<Expression>  := <Field> | <Constant>
<Term>        := <Expression> = <Expression>
<Predicate>   := <Term> [ AND <Predicate> ]

<Query>       := SELECT <SelectList> FROM <TableList>
                 [ WHERE <Predicate> ]
<SelectList>  := <Field> [ , <SelectList> ]
<TableList>   := IdTok [ , <TableList> ]

<UpdateCmd>   := <Insert> | <Delete> | <Modify> | <Create>
<Create>      := <CreateTable> | <CreateView>
                 | <CreateIndex>
<Insert>      := INSERT INTO IdTok ( <FieldList> )
                 VALUES ( <ConstList> )
<FieldList>   := <Field> [ , <FieldList> ]
<ConstList>   := <Constant> [ , <Constant> ]

<Delete>      := DELETE FROM IdTok [ WHERE <Predicate> ]

<Modify>      := UPDATE IdTok SET <Field> = <Expression>
                 [ WHERE <Predicate> ]

<CreateTable> := CREATE TABLE IdTok ( <FieldDefs> )
<FieldDefs>   := <FieldDef> [ , <FieldDefs> ]
<FieldDef>    := IdTok <TypeDef>
<TypeDef>     := INT | VARCHAR ( IntTok )

<CreateView>  := CREATE VIEW IdTok AS <Query>

<CreateIndex> := CREATE INDEX IdTok ON IdTok ( <Field> )
```

Figure 18-4
The grammar for the SimpleDB subset of SQL

Sözdizim (Çözümleme)-2) Özyineli İniş Çözümleme

- **<u>ÇÖZÜMLEME AĞACI</u>**: SQL cümlesinin, iç düğümlerin sözdizim kategorilerine, yaprakların ise öğelere denk geldiği ağaçtır.
- Şayet «SQL cümlesi ➔Çözümleme ağacı» ise SQL cümlesi sözdizimine uygundur.

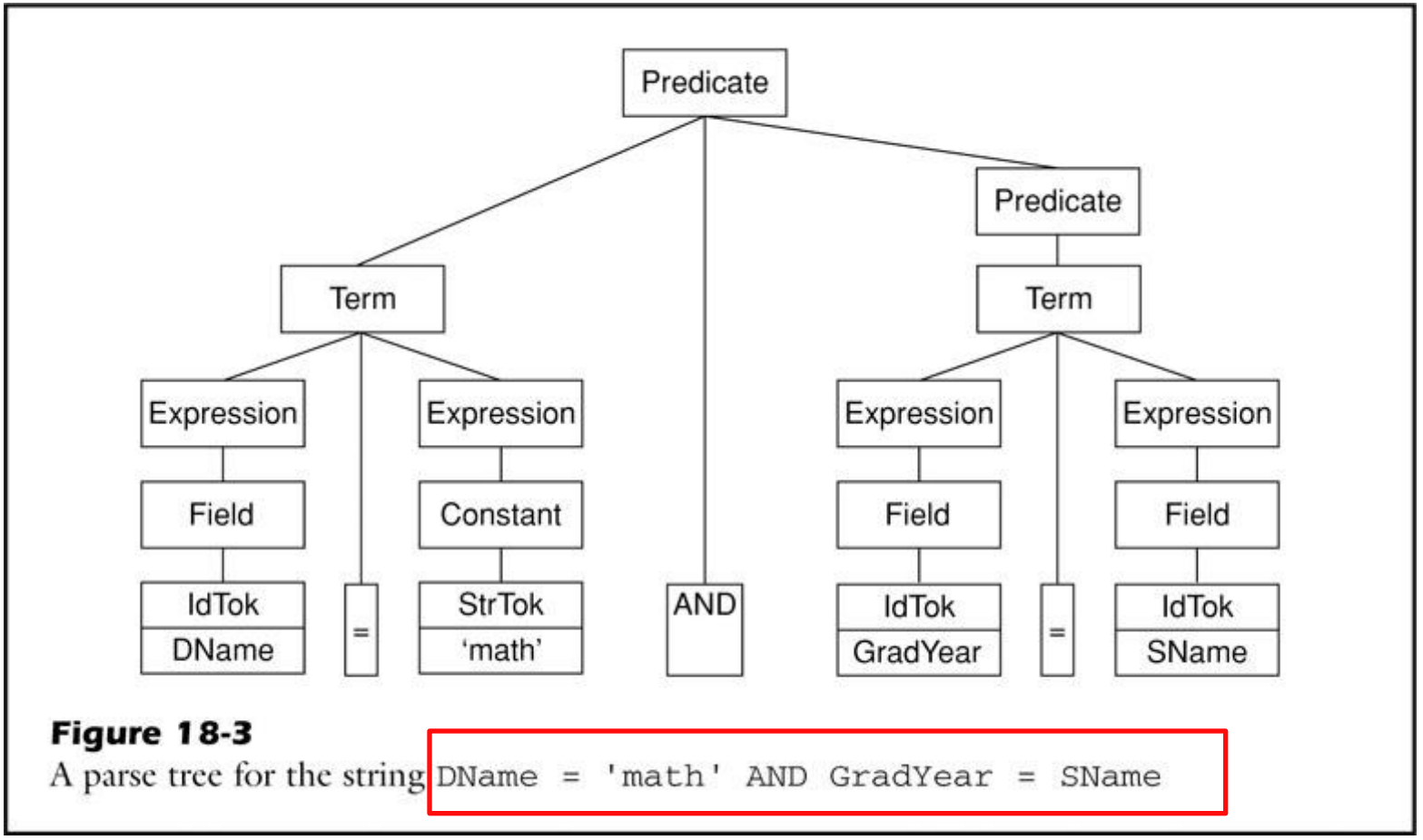


Figure 18-3
A parse tree for the string `DName = 'math' AND GradYear = SName`

Sözdizim (Çözümleme)-2) Özyineli İniş Çözümleme

- İlk 5 kategoriye ait olan gerçekleme örneği
- Her bir kategori bir fonksiyon ile gerçeklenir. Eğer gelen öğe tüketilebiliyorsa (eat….() fonksiyonundan hatasız dönüyorsa) «String s» çözümlenebilecek.
- Kategoriye ait fonksiyonların çağırılma sırası çözümleme ağacını belirler.

```
public class PredParser {
    private Lexer lex;

    public PredParser(String s) {
        lex = new Lexer(s);
    }

    public void field() {
        lex.eatId();
    }

    public void constant() {
        if (lex.matchStringConstant())
            lex.eatStringConstant();
        else
            lex.eatIntConstant();
    }

    public void expression() {
        if (lex.matchId())
            field();
        else
            constant();
    }

    public void term() {
        expression();
        lex.eatDelim('=');
        expression();
    }

    public void predicate() {
        term();
        if (lex.matchKeyword("and")) {
            lex.keyword("and");
            predicate();
        }
    }
```

SimpleDB'de Çözümleme

- SQL cümlesi çözümlenirken bazı bilgilerin toplanması gerekiyor. **Planner** sınıfı, çözümleyici (*Parser*) çağırır ve toplanan bilgileri kullanır.

SQL ifadesi	Toplanan bilgi
Sorgu (*query*)	select kısmındaki nitelik isimleri,from kısmındaki tablo isimleri ve where kısmındaki yüklemler.
Ekleme (*insert*):	tablo ismi, nitelik isim listesi ve değer listesi
Silme (*delete*):	tablo ismi ve yüklem
Yenileme (*update*):	tablo ismi, değişen nitelik ismi, yeni değer için expression, yüklem
Tablo oluşturma (*create table*)	Tablo ismi ve şeması
Görüntü oluşturma (*create view*)	Görüntü ismi ve tanımı (*definition*)
İndeks oluşturma (*create index*)	İndeks ismi, tablo ismi, indekslenen nitelik ismi

SimpleDB'de Çözümleme- Query kısmı

```
public class Parser {
   private Lexer lex;

   public Parser(String s) {
      lex = new Lexer(s);
   }

// Methods for parsing predicates, terms, etc.

   public String field() {
      return lex.eatId();
   }

   public Constant constant() {
      if (lex.matchStringConstant())
         return new StringConstant
                              (lex.eatStringConstant());
      else
         return new IntConstant(lex.eatIntConstant());
   }

   public Expression expression() {
      if (lex.matchId())
         return new FieldNameExpression(field());
      else
         return new ConstantExpression(constant());
   }
```

Figure 18-6
The code for the SimpleDB class *Parser*

```
      lex.eatDelim('=');
      Expression rhs = expression();
      return new Term(lhs, rhs);
   }

   public Predicate predicate() {
      Predicate pred = new Predicate(term());
      if (lex.matchKeyword("and")) {
         lex.eatKeyword("and");
         pred.conjoinWith(predicate());
      }
      return pred;
   }

// Methods for parsing queries

   public QueryData query() {
      lex.eatKeyword("select");
      Collection<String> fields = selectList();
      lex.eatKeyword("from");
      Collection<String> tables = tableList();
      Predicate pred = new Predicate();
      if (lex.matchKeyword("where")) {
         lex.eatKeyword("where");
         pred = predicate();
      }
      return new QueryData(fields, tables, pred);
   }

   private Collection<String> selectList() {
      Collection<String> L = new ArrayList<String>();
      L.add(field());
      if (lex.matchDelim(',')) {
         lex.eatDelim(',');
         L.addAll(selectList());
      }
      return L;
   }

   private Collection<String> tableList() {
      Collection<String> L = new ArrayList<String>();
      L.add(lex.eatId());
      if (lex.matchDelim(',')) {
         lex.eatDelim(',');
         L.addAll(tableList());
      }
      return L;
   }
```

SimpleDB'de Çözümleme-Query kısmı

```
public class QueryData {
   private Collection<String> fields;
   private Collection<String> tables;
   private Predicate pred;

   public QueryData(Collection<String> fields,
         Collection<String> tables, Predicate pred) {
      this.fields = fields;
      this.tables = tables;
      this.pred = pred;
   }

   public Collection<String> fields() {
      return fields;
   }

   public Collection<String> tables() {
      return tables;
   }

   public Predicate pred() {
      return pred;
   }

   public String toString() {
      String result = "select ";
      for (String fldname : fields)
         result += fldname + ", ";

      //remove final comma
      result = result.substring(0, result.length()-2);
      result += " from ";
      for (String tblname : tables)
         result += tblname + ", ";
      result = result.substring(0, result.length()-2);
      String predstring = pred.toString();
      if (!predstring.equals(""))
         result += " where " + predstring;
      return result;
   }
}
```

SimpleDB’de Çözümleme

- Parser’ın diğer kısımları benzer şekilde olup; gerçekleme kitaptan takip edilebilir…