Introduction to VanillaDB

DB/AI Bootcamp

2018 Summer

DataLab, CS, NTHU

RDBMS

- Definition: A ***Relational DBMS*** (***RDBMS***) is a DBMS that supports the relational model

Outline

- Architecture of an RDBMS
- Query interfaces
  - SQL, JDBC, and native interface
- Storage interface
  - RecordFile and metadata

Outline

- Architecture of an RDBMS
- Query interfaces
  - SQL, JDBC, and native interface
- Storage interface
  - RecordFile and metadata

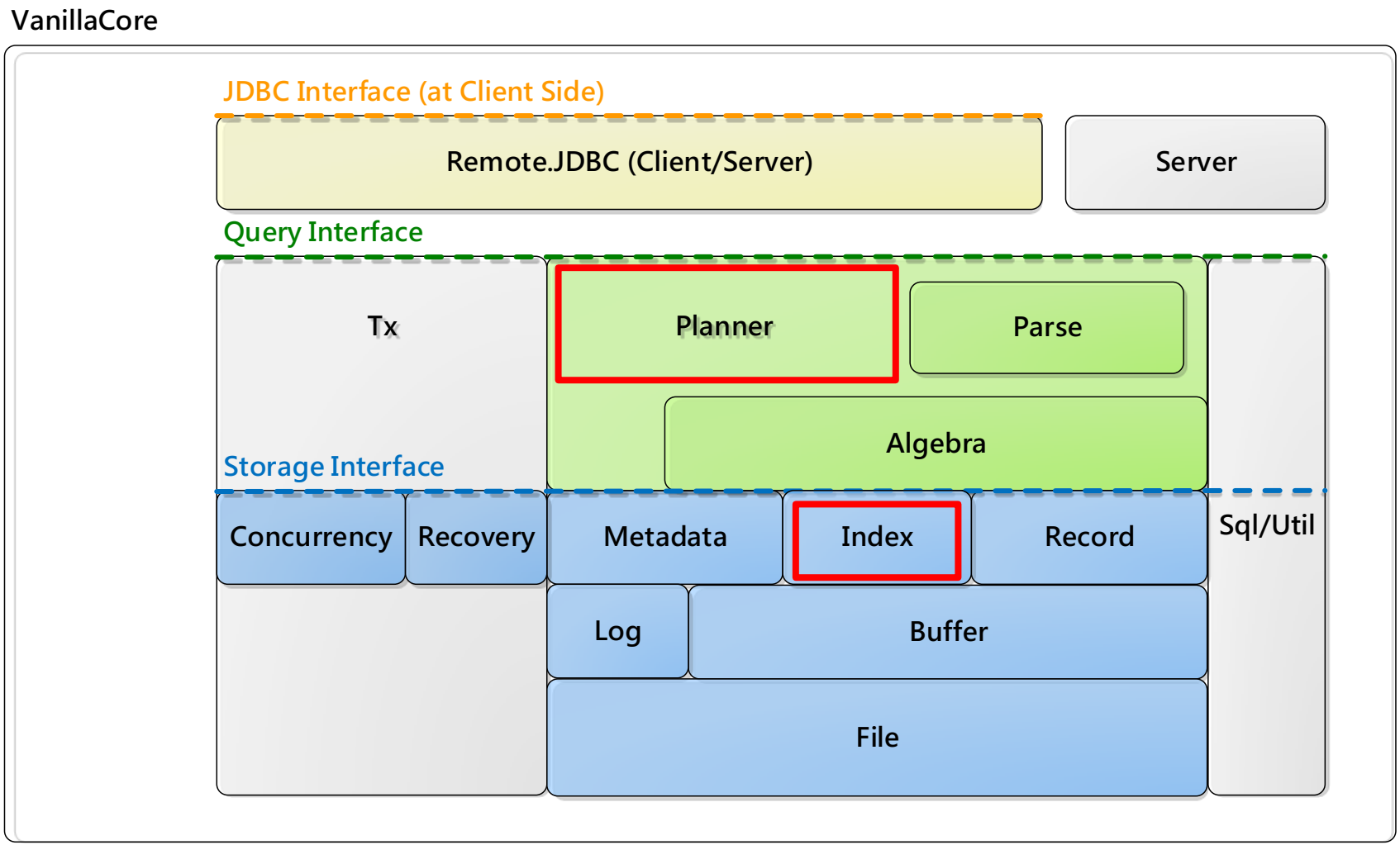
Architecture of an RDBMS

- Largely influenced by the IBM System R
  - Announced in 1974

The VanillaDB Project

- VanillaCore
  - An RDBMS that runs on a single server
- VanillaComm
  - A communication infrastructure for distributed RDBMS

Architecture of VanillaCore (1/2)



Architecture of VanillaCore (2/2)

- Interfaces:
  - SQL
  - JDBC
  - Native query interface
  - Storage interface (for file access)
- Key components:
  - Sever and infrastructures (jdbc, sql, tx, and utils)
  - Query engine
  - Storage engine

Outline

- Architecture of an RDBMS
- Query interfaces
  - SQL, JDBC, and Native
- Storage interface
  - RecordFile and metadata

The SQL Interface

- ***SQL*** (Structured Query Language) is a standardized interface
  - SQL-92, SQL-99, and later versions

Issuing SQL Commands

- Client-server mode:
  - Manual commands through `util.ConsoleSQLInterpreter`

```
ConsoleSQLInterpreter [Java Application] C:\Program Files\Java\jdk1.6.0_27\bin\javaw.exe (2013/2/4 上午12:29:52)

SQL> SELECT sname FROM student WHERE gradyear > 2012

     sname
----------
       dun

SQL>
```

  - Or in client programs through the ***JDBC*** interface
- Embedded mode:
  - Through the native query interface

Supported SQL Commands (1/5)

- VanillaCore supports a tiny subset of SQL-92
  - DDL: CREATE <TABLE | VIEW | INDEX>
  - DML: SELECT, UPDATE, INSERT, DELETE
- Limitations:
  - Types: int, long, double, and varchar
  - Single SELECT-FROM-WHERE block
    - No * in SELECT clause, no AS in FROM, no null value, no explicit JOIN or OUTER JOIN, only AND in WHERE, no parentheses, no computed value
  - Arithmetic expression only in UPDATE
  - No query in INSERT

Supported SQL Commands (2/5)

```
<Field>       := IdTok
<Constant>    := StrTok | NumericTok
<Expression> := <Field> | <Constant>
<BinaryArithmeticExpression>  :=
             ADD(<Expression>, <Expression>) |
             SUB(<Expression>, <Expression>) |
             MUL(<Expression>, <Expression>) |
             DIV(<Expression>, <Expression>)
<Term>        := <Expression> = <Expression> |
             <Expression> > <Expression>  |
             <Expression> >= <Expression> |
             <Expression> < <Expression>  |
             <Expression> <= <Expression>
<Predicate>  := <Term> [ AND <Predicate> ]
```

Supported SQL Commands (3/5)

```
<Query>       := SELECT <ProjectSet> FROM <TableSet>
              [ WHERE <Predicate> ] [ GROUP BY <IdSet> ]
              [ ORDER BY <SortList> [ DESC | ASC ] ]

<IdSet>       := <Field> [ , <IdSet> ]
<TableSet>    := IdTok [ , <TableSet> ]
<AggFn>       := AVG(<Field>) | COUNT(<Field>) |
              COUNT(DISTINCT <Field>) | MAX(<Field>) |
              MIN(<Field>) | SUM(<Field>)
<ProjectSet>  := <Field> | <AggFn> [ , <ProjectSet>]
<SortList>    := <Field> | <AggFn> [ , <SortList>]
```

Supported SQL Commands (4/5)

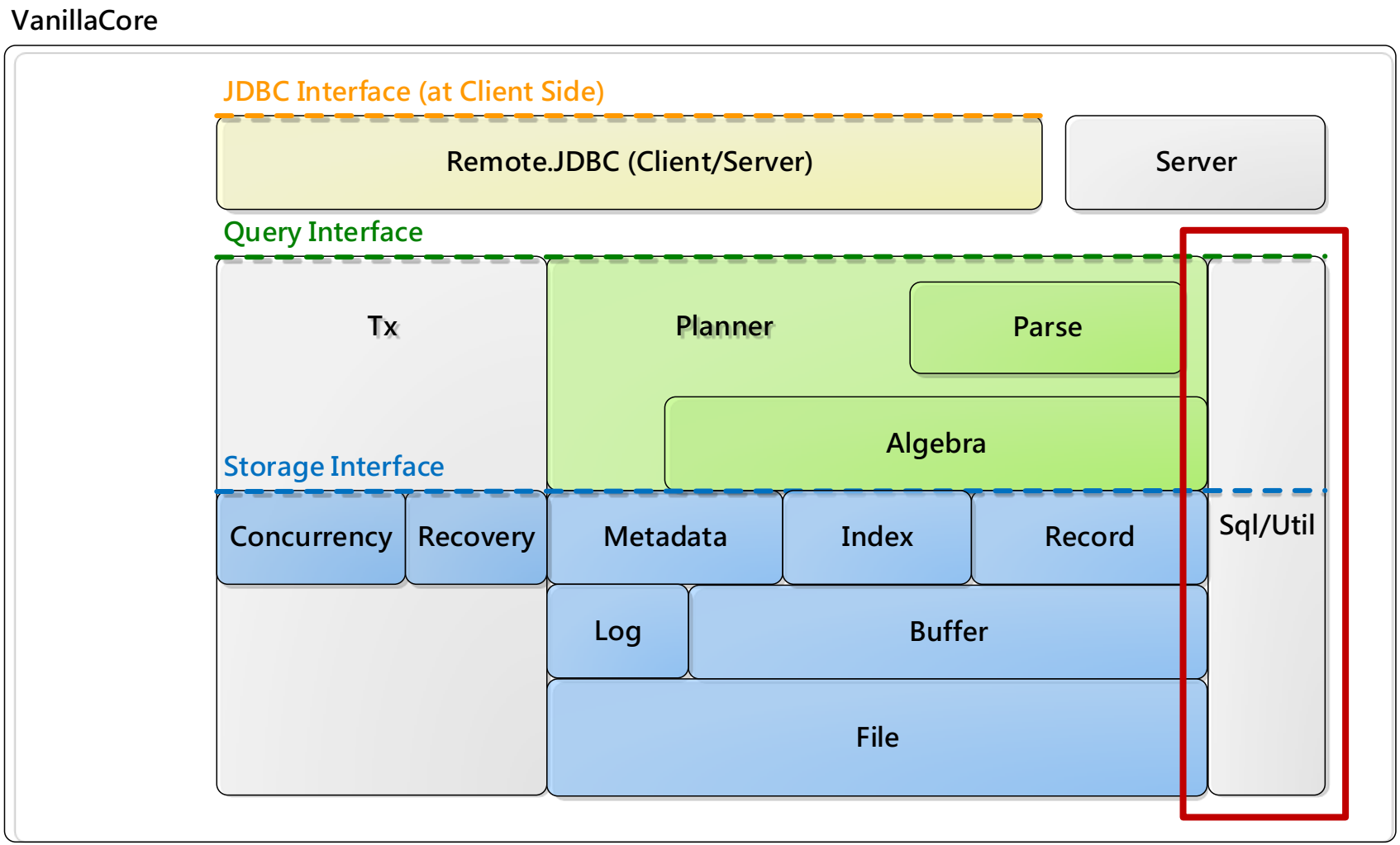
```
<UpdateCmd>   := <Insert> | <Delete> | <Modify> | <Create>
<Create>      := <CreateTable> | <CreateView> |
                    <CreateIndex>
<Insert>      := INSERT INTO IdTok ( <FieldList> ) VALUES
                    ( <ConstantList> )
<FieldList>   := <Field> [ , <Field> ]
<ConstantList>:= <Constant> [ , <Constant> ]
<Delete>      := DELETE FROM IdTok [ WHERE <Predicate> ]
<Modify>      := UPDATE IdTok SET <ModifyTermList>
                    [ WHERE <Predicate> ]
```

Supported SQL Commands (5/5)

```
<ModifyExpression>  := <Expression> |
                           <BinaryArithmeticExpression>
<ModifyTermList>    := <Field> = <ModifyExpression>
                           [ , <ModifyTermList> ]

<CreateTable>       := CREATE TABLE IdTok ( <FieldDefs> )
<FieldDefs>         := <FieldDef> [ , <FieldDef> ]
<FieldDef>          := IdTock <TypeDef>
<TypeDef>           := INT | LONG | DOUBLE |
                           VARCHAR ( NumericTok )
<CreateView>        := CREATE VIEW IdTok AS <Query>
<CreateIndex>       := CREATE INDEX IdTok ON IdTok
                           ( <Field> )
```

Architecture of VanillaCore



Utility Classes for SQL

- In `sql` package
- Types:
  - Numeric: `IntegerType`, `BigIntType`, and `DoubleType`
  - String: `VarcharType`
- Constants:
  - `IntegerConstant`, `BigIntConstant`, `DoubleConstant`, and `VarcharConstant`
- For relations:
  - `Schema`, `Record`
- For commands:
  - `Predicate`, `AggFn`

Types

- Each `Type` impl. denotes a supported SQL type

```
<<abstract>>
Type
---
<<final>> + INTEGER : Type
<<final>> + BIGINT : Type
<<final>> + DOUBLE : Type
<<final>> + VARCHAR : Type
---
+ VARCHAR(arg : int) : Type
+ newInstance(sqlType : int) : Type
+ newInstance(sqlType : int, arg : int) : Type
<<abstract>> + getSqlType() : int
<<abstract>> + getArgument() : int
<<abstract>> + isFixedSize() : boolean
<<abstract>> + isNumeric() : boolean
<<abstract>> + maxSize() : int
<<abstract>> + maxValue() : Constant
<<abstract>> + minValue() : Constant
```

java.sql.Types	vanilladb.sql.Type
INTEGER	IntegerType
BIGINT	BigIntType
DOUBLE	DoubleType
VARCHAR	VarcharType

Constants

- Each `Constant` impl. denotes a value of a supported type
  - Immutable
  - Arithmetics with auto type-upgrade

```
<<abstract>>
Constant
---
---
+ newInstance(type : Type, val : byte[]) :
        Constant
+ defaultInstance(type : Type) : Constant
<<abstract>> + getType() : Type
<<abstract>> + asJavaVal() : Object
<<abstract>> + asBytes() : byte[]
<<abstract>> + size() : int
<<abstract>> + castTo(type : Type) : Constant
<<abstract>> + add(c : Constant) : Constant
<<abstract>> + sub(c : Constant) : Constant
<<abstract>> + mul(c : Constant) : Constant
<<abstract>> + div(c : Constant) : Constant
```

vanilladb.sql.Type	Value type in Java
IntegerType	Integer
BigIntType	Long
DoubleType	Double
VarcharType	String

Relations

blog_pages

blog_id	url	created	author_id
33981	ms.com/…	2012/10/31	729
33982	apache.org/…	2012/11/15	4412

Schema

Record

Schema & Record

```
Schema
---
+ Schema()
+ addField(fldname : String, type : Type)
+ add(fldname : String, sch : Schema)
+ addAll(sch : Schema)
+ fields() : SortedSet<String>
+ hasField(fldname : String) : boolean
+ type(fldname : String) : Type
```

- Contains the name and type of each field in a table

```
<<interface>>
Record
---
+ getVal(fldName : String) : Constant
```

- A map from field names to constants

Commands

- Supporting WHERE: predicates in `sql.predicate` package
  - `Expression`, `FieldExpression`, `ConstantExpression`, `BinaryArithmeticExpression`, `Term`, and `Predicate`
- Supporting GROUP BY: aggregation functions in the `sql.aggfn` package
  - `AggregationFn`, `AvgFn`, `CountFn`, `DistictCountFn`, `MaxFn`, `MinFn` and `SumFn`

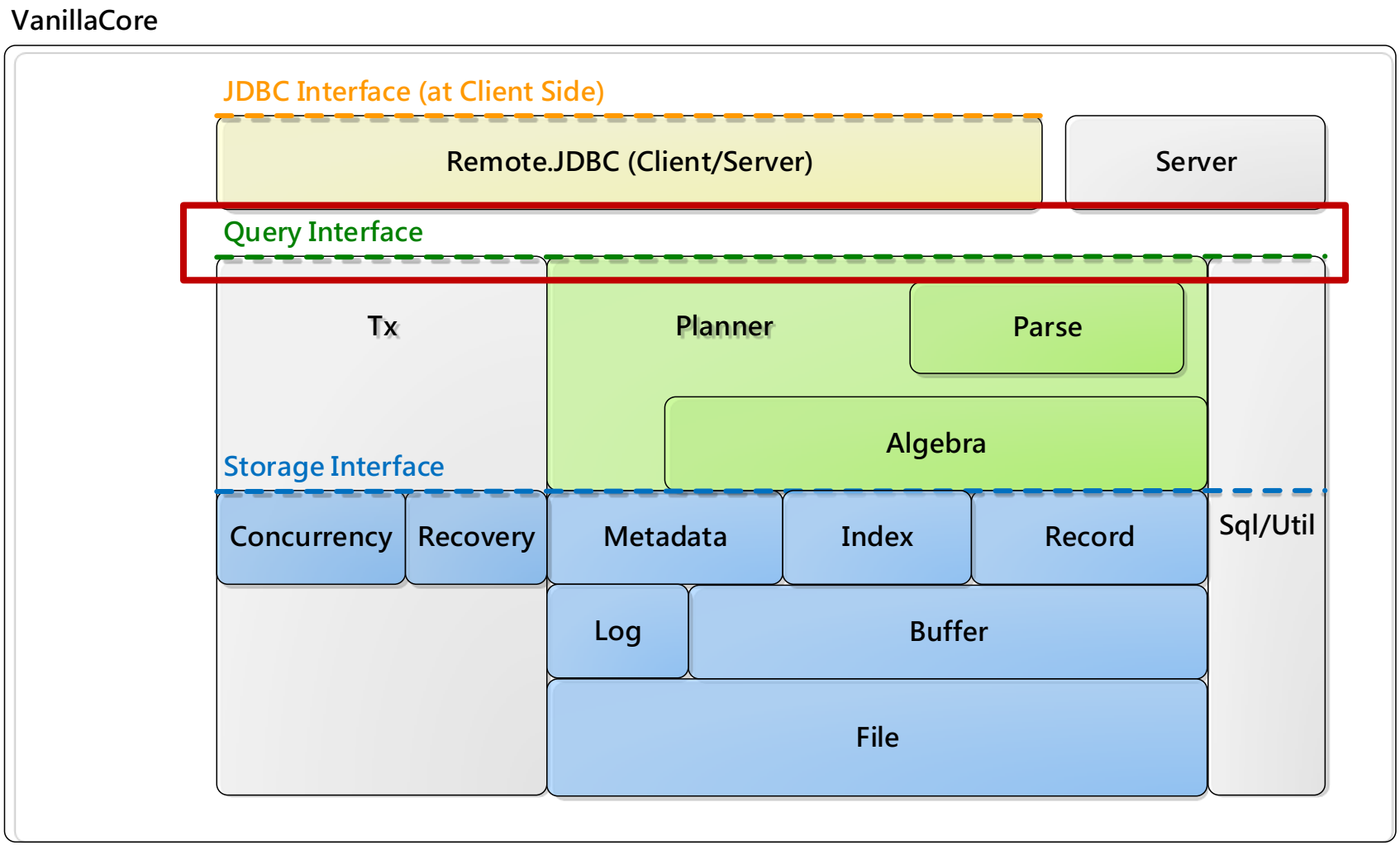
Outline

- Architecture of an RDBMS
- Query interfaces
  - SQL, JDBC, and native interface
- Storage interface
  - RecordFile and metadata

Outline

- Architecture of an RDBMS
- Query interfaces
  - SQL, JDBC, and native interface
- Storage interface
  - RecordFile and metadata

Architecture of VanillaCore (1/2)



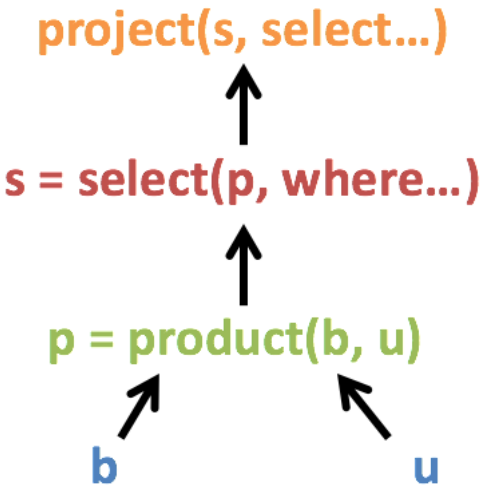
API (1/2)

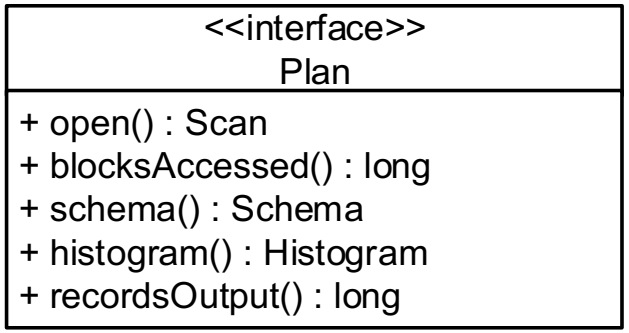
Planner
+ createQueryPlan(qry : String, tx : Transaction) : Plan + executeUpdate(cmd : String, tx : Transaction) : int

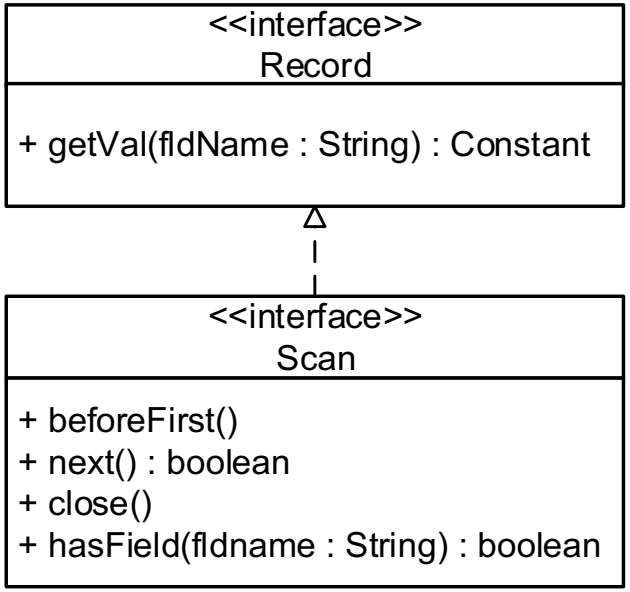
Transaction
<u>+ addStartListener (l : TransactionLifeCycleListener)</u> + Transaction(concurMgr : TransactionLifeCycleListener, recoveryMgr : TransactionLifeCycleListener, readOnly : boolean, txNum : long) + addLifeCycleListener(l : TransactionLifeCycleListener) + commit() + rollback() + recover() + endStatement() + getTransactionNumber() : long + isReadOnly() : boolean + concurrencyMgr() : ConcurrencyMgr + recoveryMgr() : RecoveryMgr

- All operations resulted from a planner are bound by the associated tx

API (2/2)







- Corresponds to an operator in relational algebra
  - The root of a plan tree
  - For ***cost estimation*** only
  - `open()` propagates down to the tree
- Iterator of output records of a partial query
  - Actual data access

References

- M.M. Astrahan et al., System R: relational approach to database management, *ACM Transactions on Database Systems*, Vol. 1, No. 2, 1976
- J. M. Hellerstein et al., Architecture of a database system, *Foundations and Trends in Databases*, Vol. 1, No. 2, 2007
- Edward Sciore, Chapters 8 & 20, *Database Design and Implementation*, 2008