Metadata Management

vanilladb.org

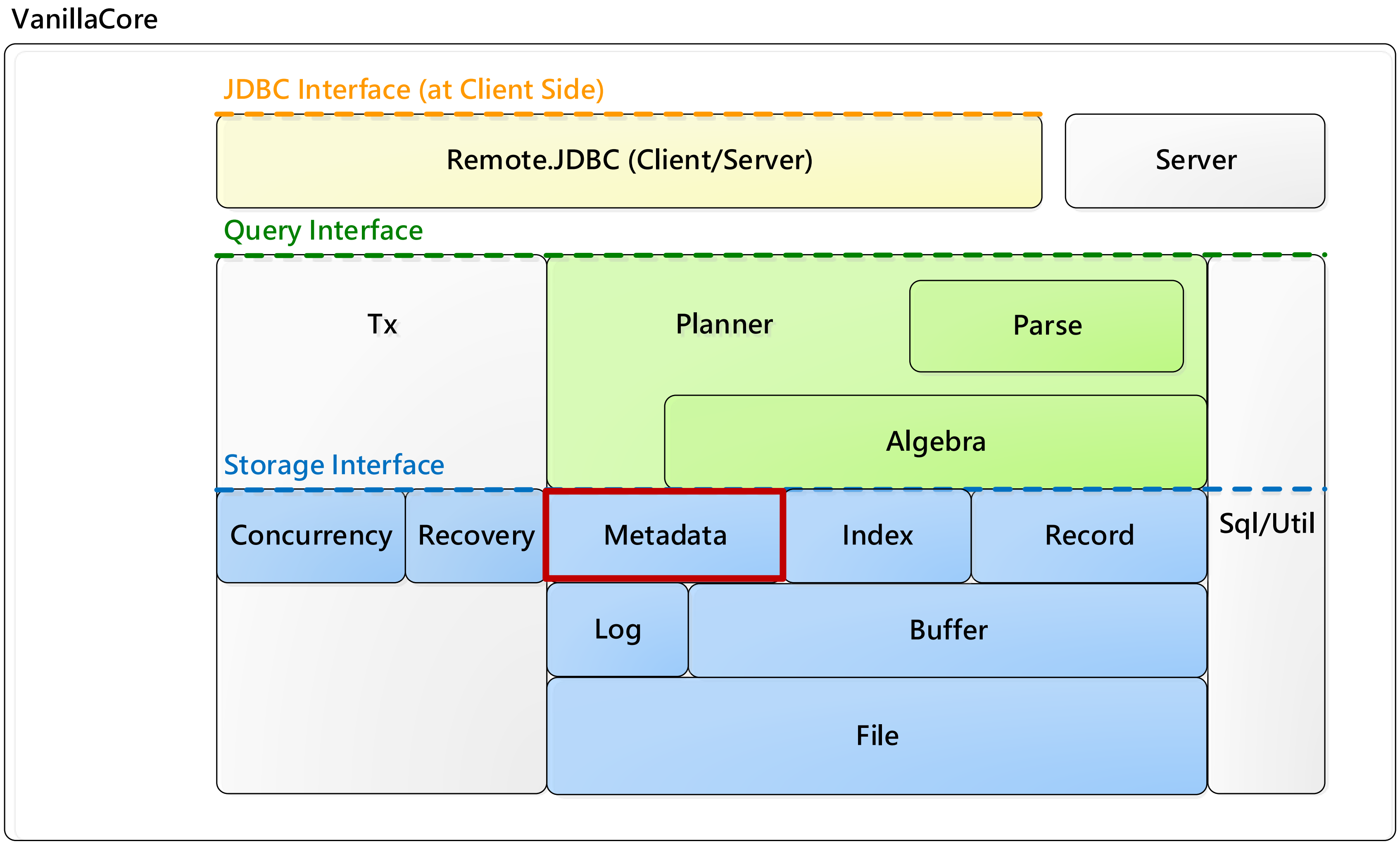
Outline

- Overview
- Managing Table Metadata
- Managing View Metadata
- Managing Statistical Metadata
- Implementing VanillaCore Metadata Manager

Overview

- A ***metadata*** is the information about a database, apart from its contents
- In previous lecture (see topic 3, Architecture and Interfaces), we have seen several kinds of metadata in VanillaCore
- Now we are going to examine the implementation of ***metadata manager***

Architecture of VanillaCore (1/2)



Metadata in VanillaCore

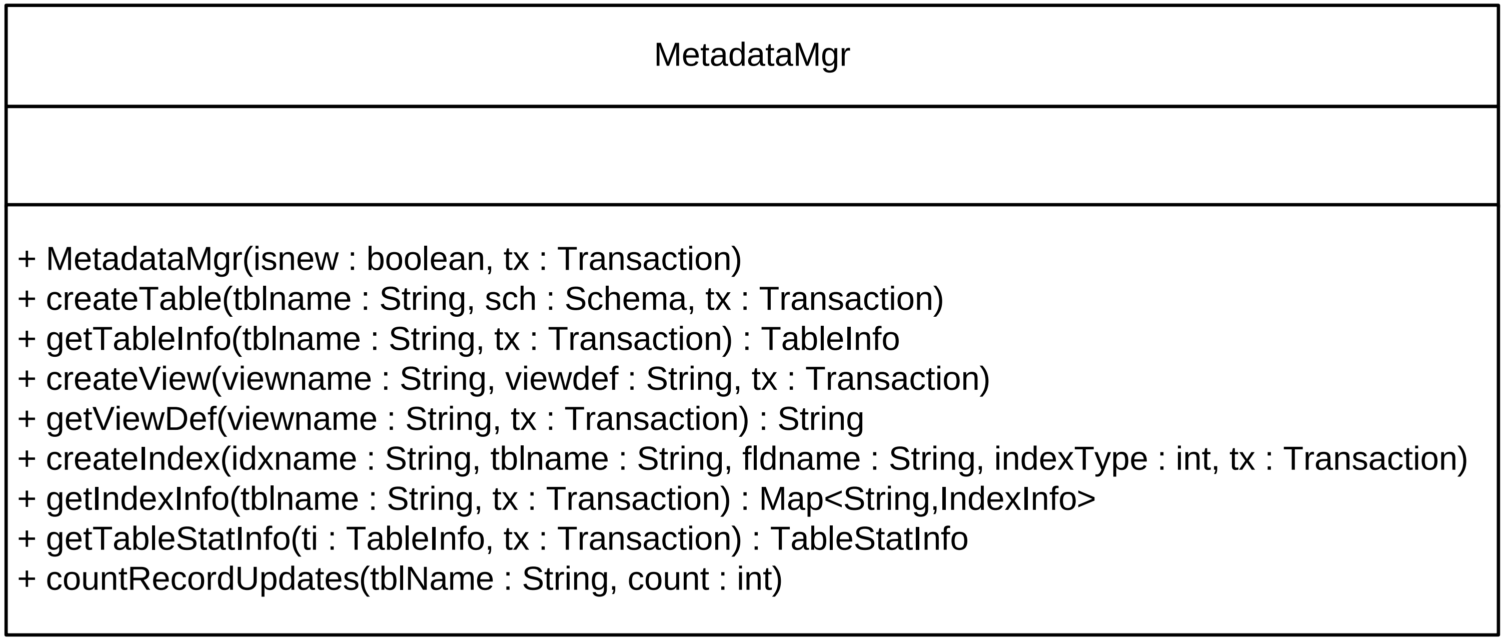
- Table metadata
  - Describes the file of each table, and structure of the table's records such as the length, type, and offset of each field
- View metadata
  - Describes the properties of each view, such as its definition and creator
- Index metadata
  - Describes the indexes that have been defined on each field
- Statistical metadata
  - Describes the statistics of each table useful to estimating the cost of plan tree

Metadata in Database System

- VanillaCore stores the first three types of metadata in a collection of special tables called the ***catalog tables***
  - Allows the metadata to be queried like normal data
- Statistical metadata is kept in memory and updated periodically
  - No need to be accurate
  - Accessed by every plan tree, must be very fast

Metadata Manager

- The storage engine provides an ***metadata manager***
  - Single instance is created when system startup
  - Client can get it from the method `mdMgr` in `VanillaDB`



```
MetadataMgr

+ MetadataMgr(isnew : boolean, tx : Transaction)
+ createTable(tblname : String, sch : Schema, tx : Transaction)
+ getTableInfo(tblname : String, tx : Transaction) : TableInfo
+ createView(viewname : String, viewdef : String, tx : Transaction)
+ getViewDef(viewname : String, tx : Transaction) : String
+ createIndex(idxname : String, tblname : String, fldname : String, indexType : int, tx : Transaction)
+ getIndexInfo(tblname : String, tx : Transaction) : Map<String,IndexInfo>
+ getTableStatInfo(ti : TableInfo, tx : Transaction) : TableStatInfo
+ countRecordUpdates(tblName : String, count : int)
```

Outline

- Overview
- Managing Table Metadata
- Managing View Metadata
- Managing Statistical Metadata
- Implementing VanillaCore Metadata Manager

Table Metadata Manager

- Stores the table information when creating table
  - Clients call the method `createTable`
- Provides table information
  - Clients can get the table information through the method `getTableInfo`
- The `TableInfo` object stores a table's information

TableInfo

```
public class TableInfo {
      private Schema schema;
      private Map<String, Integer> offsets;
      private int recSize;
      private String tblName;

      public TableInfo(String tblName, Schema schema) {
            this.schema = schema;
            this.tblName = tblName;
            offsets = new HashMap<String, Integer>();
            int pos = 0;
            for (String fldName : schema.fields()) {
                  offsets.put(fldName, pos);
                  pos += Page.maxSize(schema.type(fldName));
            }
            recSize = pos < minRecSize ? minRecSize : pos;
      }

      public TableInfo(String tblName, Schema schema,
                  Map<String, Integer> offsets, int recSize) {
            this.tblName = tblName;
            this.schema = schema;
            this.offsets = offsets;
            this.recSize = recSize;
      }

      public String fileName() {
            return tblName + ".tbl";
      }

      public String tableName() {
            return tblName;
      }

      public Schema schema() {
            return schema;
      }

      public int offset(String fldName) {
            return offsets.get(fldName);
      }

      public int recordSize() {
            return recSize;
      }

      public RecordFile open(Transaction tx) {
            return new RecordFile(this, tx);
      }
}
```

Implementing Table Metadata

- How to store the table information in system catalog table?
- VanillaCore hold its table metadata in two catalog tables
  - `tblcat(tblname:varchar(MAX_NAME), recsize:integer)`
  - `fldcat(tblname:varchar(MAX_NAME), fldname:varchar(MAX_NAME), type:integer, typearg:integer, offset:integer)`

Implementing Table Metadata

- There is one record in the `tblcat` table for each database table
- There is one record in the `fldcat` table for each field of each table
- The catalog tables also contain records that encode their own metadata

tblname	recsize
students	60
departments	36
courses	42
section	30
…	…

Values defined in java.sql.type

tblname	fldname	type	typearg	offset
students	s-id	4	0	0
students	s-name	12	20	4
students	major-id	4	0	64
students	grad-year	-5	0	68
departments	d-id	4	0	0
departments	d-name	12	20	4
…	…	…	…	…

Implementing Table Metadata

- The class `TableMgr` implements the table-metadata methods
  - The constructor is called during system start-up
    - Creates the schemas for `tblcat` and `fldcat` and calculates their `TableInfo` object
    - If the database is new, creates these two tables

TableMgr

```
public class TableMgr {
	public static final String TCAT = "tblcat";
	public static final String TCAT_TBLNAME = "tblname",
			TCAT_RECSIZE = "recsize";
	public static final String FCAT = "fldcat";
	public static final String FCAT_TBLNAME = "tblname",
			FCAT_FLDNAME = "fldname", FCAT_TYPE = "type",
			FCAT_TYPEARG = "typearg", FCAT_OFFES = "offset";
	public static final int MAX_NAME;
	private TableInfo tcatInfo, fcatInfo;

	public TableMgr(boolean isNew, Transaction tx) {
		Schema tcatSchema = new Schema();
		tcatSchema.addField(TCAT_TBLNAME, VARCHAR(MAX_NAME));
		tcatSchema.addField(TCAT_RECSIZE, INTEGER);
		tcatInfo = new TableInfo(TCAT, tcatSchema);

		Schema fcatSchema = new Schema();
		fcatSchema.addField(FCAT_TBLNAME, VARCHAR(MAX_NAME));
		fcatSchema.addField(FCAT_FLDNAME, VARCHAR(MAX_NAME));
		fcatSchema.addField(FCAT_TYPE, INTEGER);
		fcatSchema.addField(FCAT_TYPEARG, INTEGER);
		fcatSchema.addField(FCAT_OFFES, INTEGER);
		fcatInfo = new TableInfo(FCAT, fcatSchema);

		if (isNew) {
			formatFileHeader(TCAT, tx);
			formatFileHeader(FCAT, tx);
			createTable(TCAT, tcatSchema, tx);
			createTable(FCAT, fcatSchema, tx);
		}
	}

	public void createTable(String tblName, Schema sch, Transaction tx) {
		if (tblName != TCAT_TBLNAME && tblName != FCAT_TBLNAME)
			formatFileHeader(tblName, tx);
		TableInfo ti = new TableInfo(tblName, sch);
		// insert one record into tblcat
		RecordFile tcatfile = tcatInfo.open(tx);
		tcatfile.insert();
		tcatfile.setVal(TCAT_TBLNAME, new VarcharConstant(tblName));
		tcatfile.setVal(TCAT_RECSIZE, new IntegerConstant(ti.recordSize()));
		tcatfile.close();

		// insert a record into fldcat for each field
		RecordFile fcatfile = fcatInfo.open(tx);
		for (String fldname : sch.fields()) {
			fcatfile.insert();
			fcatfile.setVal(FCAT_TBLNAME, new VarcharConstant(tblName));
			fcatfile.setVal(FCAT_FLDNAME, new VarcharConstant(fldname));
			fcatfile.setVal(FCAT_TYPE, new IntegerConstant(sch.type(fldname)
					.getSqlType()));
			fcatfile.setVal(FCAT_TYPEARG, new IntegerConstant(sch.type(fldname)
					.getArgument()));
			fcatfile.setVal(FCAT_OFFES, new IntegerConstant(ti.offset(fldname)));
		}
		fcatfile.close();
	}
```

TableMgr

```
    public TableInfo getTableInfo(String tblName, Transaction tx) {
        RecordFile tcatfile = tcatInfo.open(tx);
        tcatfile.beforeFirst();
        int recsize = -1;
        while (tcatfile.next()) {
            String t = (String) tcatfile.getVal(TCAT_TBLNAME).asJavaVal();
            if (t.equals(tblName)) {
                recsize = (Integer) tcatfile.getVal(TCAT_RECSIZE).asJavaVal();
                break;
            }
        }
        tcatfile.close();

        RecordFile fcatfile = fcatInfo.open(tx);
        fcatfile.beforeFirst();
        Schema sch = new Schema();
        Map<String, Integer> offsets = new HashMap<String, Integer>();
        while (fcatfile.next())
            if (((String) fcatfile.getVal(FCAT_TBLNAME).asJavaVal())
                    .equals(tblName)) {
                String fldname = (String) fcatfile.getVal(FCAT_FLDNAME)
                        .asJavaVal();
                int fldtype = (Integer) fcatfile.getVal(FCAT_TYPE).asJavaVal();
                int fldarg = (Integer) fcatfile.getVal(FCAT_TYPEARG)
                        .asJavaVal();
                int offset = (Integer) fcatfile.getVal(FCAT_OFFES).asJavaVal();
                offsets.put(fldname, offset);
                sch.addField(fldname, Type.newInstance(fldtype, fldarg));
            }
        fcatfile.close();
        if (recsize == -1)
            return null;
        return new TableInfo(tblName, sch, offsets, recsize);
    }

    private void formatFileHeader(String tblName, Transaction tx) {
        String fileName = tblName + ".tbl";
        RecordFile.formatFileHeader(fileName, tx);
    }
}
```

Outline

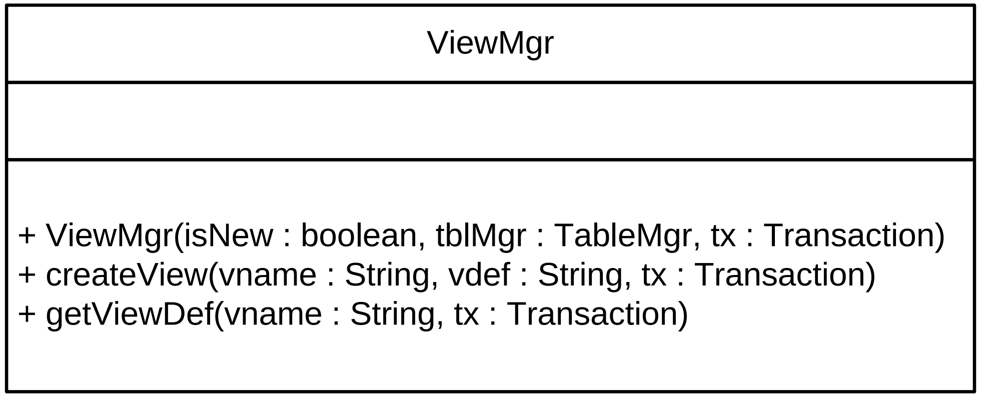
View Metadata

- A ***view*** is a table whose records are computed dynamically from a query
- That query is called the ***definition*** of the view
- The metadata manager stores the definition of each newly created view into catalog
  - `viewcat(viewname:varchar(MAX_NAME), viewdef:varchar(MAX_VIEWDEF)`

Viewname	viewdef
all-students-name	select sname from students
…	…

View Metadata Manager

- The class `ViewMgr` is responsible for storing/ retrieving the view definition
- The constructor is called during system start-up and creates the `viewcat` table if the database is new


ViewMgr

```
+ ViewMgr(isNew : boolean, tblMgr : TableMgr, tx : Transaction)
+ createView(vname : String, vdef : String, tx : Transaction)
+ getViewDef(vname : String, tx : Transaction)
```

ViewMgr

```
class ViewMgr {
    public static final String VCAT = "viewcat";
    public static final String VCAT_VNAME = "viewname", VCAT_VDEF = "viewdef";
    private static final int MAX_VIEWDEF;
    TableMgr tblMgr;
    public ViewMgr(boolean isNew, TableMgr tblMgr, Transaction tx) {
        this.tblMgr = tblMgr;
        if (isNew) {
            Schema sch = new Schema();
            sch.addField(VCAT_VNAME, VARCHAR(MAX_NAME));
            sch.addField(VCAT_VDEF, VARCHAR(MAX_VIEWDEF));
            tblMgr.createTable(VCAT, sch, tx);
        }
    }

    public void createView(String vName, String vDef, Transaction tx) {
        TableInfo ti = tblMgr.getTableInfo(VCAT, tx);
        RecordFile rf = ti.open(tx);
        rf.insert();
        rf.setVal(VCAT_VNAME, new VarcharConstant(vName));
        rf.setVal(VCAT_VDEF, new VarcharConstant(vDef));
        rf.close();
    }

    public String getViewDef(String vName, Transaction tx) {
        String result = null;
        TableInfo ti = tblMgr.getTableInfo(VCAT, tx);
        RecordFile rf = ti.open(tx);
        rf.beforeFirst();
        while (rf.next())
            if (((String) rf.getVal(VCAT_VNAME).asJavaVal()).equals(vName)) {
                result = (String) rf.getVal(VCAT_VDEF).asJavaVal();
                break;
            }
        rf.close();
        return result;
    }
}
```

Outline

- Overview
- Managing Table Metadata
- Managing View Metadata
- Managing Statistical Metadata
- Implementing VanillaCore Metadata Manager

Statistical Metadata

- TBA

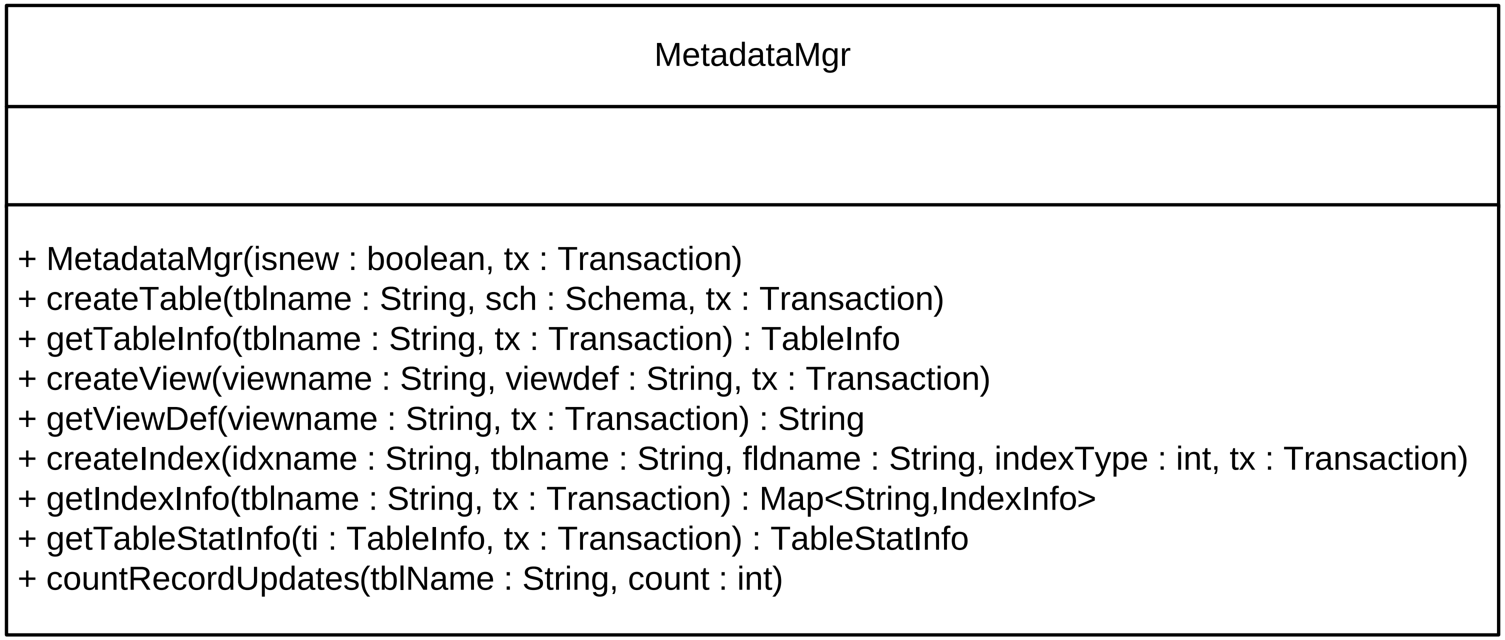
Outline

- Overview
- Managing Table Metadata
- Managing View Metadata
- Managing Statistical Metadata
- Implementing VanillaCore Metadata Manager

Implementing VanillaCore Metadata Manager

- Metadata manager is implemented via the four manager classes
  - `TableMgr`, `ViewMgr`, `StatMgr`, and `IndexMgr`
- The class `MetadataMgr` hides this distinction and provides all kinds of API to access different metadata

MetadataMgr



MetadataMgr

+ MetadataMgr(isnew : boolean, tx : Transaction)
+ createTable(tblname : String, sch : Schema, tx : Transaction)
+ getTableInfo(tblname : String, tx : Transaction) : TableInfo
+ createView(viewname : String, viewdef : String, tx : Transaction)
+ getViewDef(viewname : String, tx : Transaction) : String
+ createIndex(idxname : String, tblname : String, fldname : String, indexType : int, tx : Transaction)
+ getIndexInfo(tblname : String, tx : Transaction) : Map<String,IndexInfo>
+ getTableStatInfo(ti : TableInfo, tx : Transaction) : TableStatInfo
+ countRecordUpdates(tblName : String, count : int)

MetadataMgr

```
public class MetadataMgr {
    private static TableMgr tblMgr;
    private static ViewMgr viewMgr;
    private static StatMgr statMgr;
    private static IndexMgr idxMgr;

    public MetadataMgr(boolean isNew, Transaction tx) {
        tblMgr = new TableMgr(isNew, tx);
        viewMgr = new ViewMgr(isNew, tblMgr, tx);
        idxMgr = new IndexMgr(isNew, tblMgr, tx);
        statMgr = new StatMgr(tblMgr, tx);
    }

    public void createTable(String tblName, Schema sch, Transaction tx) {
        tblMgr.createTable(tblName, sch, tx);
    }

    public TableInfo getTableInfo(String tblName, Transaction tx) {
        return tblMgr.getTableInfo(tblName, tx);
    }

    public void createView(String viewName, String viewDef, Transaction tx) {
        viewMgr.createView(viewName, viewDef, tx);
    }

    public String getViewDef(String viewName, Transaction tx) {
        return viewMgr.getViewDef(viewName, tx);
    }

    public void createIndex(String idxName, String tblName, String fldName,
            int indexType, Transaction tx) {
        idxMgr.createIndex(idxName, tblName, fldName, indexType, tx);
    }

    public Map<String, IndexInfo> getIndexInfo(String tblName, Transaction tx) {
        return idxMgr.getIndexInfo(tblName, tx);
    }

    public TableStatInfo getTableStatInfo(TableInfo ti, Transaction tx) {
        return statMgr.getTableStatInfo(ti, tx);
    }

    public void countRecordUpdates(String tblName, int count) {
        statMgr.countRecordUpdates(tblName, count);
    }
}
```

References

- Database Design and Implementation, chapter 16. Edward Sciore.