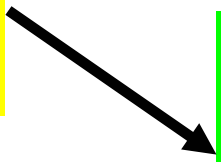
PostGreSQL database progress

- Classes that implement basic relational database interface commands with more user friendly code are in place
  - Set of abstract classes ←→ PostGres specific classes
    - PostGres provided C++ interface communicates with db via passing strings back and forth.
    - These classes define the allowed subset of SQL commands/data types/etc and handle the string conversions

MdbAbsRelDB
MdbAbsRelDBTable
MdbAbsRelDBVar
MdbAbsRelDBConstraint



MdbPostGresDB
MdbPostGresDBTable
MdbPostGresDBVar
MdbPostGresDBConstraint

- PostGres specific code is exposed to only one class (MdbDatabase)
  - All db code will need to declare a MdbDatabase object:

```
MdbDatabase mippDB;
```

Database “variables”

- Currently implemented types
  - Strings
  - Floats and 1-Dim. arrays of floats
  - Integers and 1-Dim arrays of integers
  - SQL timestamps (Month/day/year/hour/minute/second)
    - Implemented to be GMT.
- The implementation of allowed database variable types is a bit cumbersome. To create

```
MdbAbsRelDBVar* myVar=mippDB.dbVar(float xxx);
```

- To retrieve

```
myVar->value(float xxx);
```

- Templated code would have been nicer – but I couldn’t see my way through it. Suggestions?
- Other needed variable types?

Higher level table class

- Hopefully a finite number of types of tables desired. Implement generic columns and search capabilities in base classes.
- MdbTableRunSubRun class (90% of all MIPP use cases?):
  - Adds columns for min/max run, min/max subrun, and time at which row was added to the database
  - Search defined on these quantities.
  - One example table implemented so far: CKOVTestTable:

```
Class CKOVTestTable: public MdbTableRunSubRun {
Public:
  CKOVTestTable(MdbDatabase *dbFact, string dbName);
  void buildTable();
}
```

User defined columns added here

- Table name coded in constructor (ie, 1 class = 1 table)

Implementation of buildTable

- Define column names, variable types, and max array size

```
Void CKOVTestTable::buildTable() {
  for (int I; i<_nMirrors; i++) {
     std:string cN="Mirror";
    char intf(size,"%d",i);
    cN+=size;
    _colNames.push_back(cN);
    _colType.push_back(MdbAbsRelDBVar::FLOAT);
    _maxSize.push_back(1);
  }
}
```

- Other columns added on by base class.
- _colNames,_colType,_maxSize are stl vectors defined in base class.

Using CKOVTestTable

```
Int main() {
  MdbDatabase *db=new MdbDatabase();
  CKOVTestTable ckovTable(db,”test”);

 //one time only
 int ok=ckovTable.CreateTable();

//Lets add a row of random numbers
vector<MdbAbsRelDBVar*> rowtoadd;
for (i=0; i<96; i++) rowtoadd.push_back(db->dbVar((float) 0.01*i));

//Lets add this row to the table
int ok2=ckovTable.InsertRow(rowtoadd,0,0,1000);
```

=1 if PostGres interface says command succeeded

Running CKOVTestTable (Continued)

```
// What run number are we interested in.
Int run=99;

// Default is to take the most recently loaded conditions for this
//run number. Can also select on subrun if desired.

Std::vector<MdbAbsRelDBVar*> output=ckovTable.Select(run);

//output will have either 0 entries (no valid conditions), or 1
//entry per column.

cout << “Found “ << output.size() << endl;
return 0;
}
```

Already done

- Very limited SQL feature set defined. Should be easily extendable.
- Code works for my simple examples.
- In cvs (MippDatabase and CkovDBTables packages)
- SRT knows how to deal with PostGres now (at head of SRT_MIPP)

Left to do (my shortsighted list only)

- Document how to install PostGres and start up server and how to use classes
- Improve fault tolerance
- Revisit consistency of memory management
- Some technical issues to resolve with EventDataModel (=Timestamp) package / gcc3 compiling.
- Other needed features?