

H2 Database Implementation : Assignment 2

Implementing House Data Type

Kavya Kotian

Siddarth Sargunraj

Neelabh Kumar

Prashanth Kumar Purshotam

October 25, 2019

Introduction

This Assignment aims at implementing a custom Datatype called the *House* datatype. Our *House* datatype can have any values from the following list: *Condominium, Apartment, Townhome, Bungalow, Cottage, Cabin, Chalet, Barndominium, Mansion, Yurt, Castle, Palace, Chateau, Villa, Manor, Fort, Cave*.

Implementation

To implement the *HOUSE* datatype, we modify a few existing classes and also add our own classes. The modified classes are: *ErrorCode.java*, *SysProperties.java*, *TypeInfo.java*, *Value.java*. The new classes are: *CustomHouseDataType.java*, *House.java*, *valueHouse.java*. The detailed explanation of these classes are given below.

CustomHouseDataType

CustomHouseDataType implements the *CustomDataTypesHandler* interface from the package *org.h2.api*. The constructor creates an object of *Datatype* class called *houseDataType* and sets Datatype *name* to *House*, sets *type* to the ID denoted to House data type in the *Value* class and sets *sqlType* to *JAVA_OBJECT* from *java.sql.types* which is used to identify generic SQL types.

-
- `public DataType getDataTypeByName(String name)`
Checks if *name* matched our custom datatype and returns the Datatype *HOUSE*, else returns null. Given the String parameter *name* we convert it to uppercase to make it case insensitive and check if it matches out *houseDataTypeName*(i.e *HOUSE*).
 - `public DataType getDataTypeById(int type)`
Checks if *type* matches the ID denoted by *Value.HOUSE* for our custom datatype and returns the Datatype *HOUSE*, else returns null.
 - `public int getDataTypeOrder(int type)`
Returns the order denoted for the *HOUSE* class from the value class if the parameter *type* matches the order value for *HOUSE* class in the *Value* class.
 - `public Value convert(Value source, int targetType)`
Convert the provided source value into value of given target data type. You can convert a datatype to a House datatype from Object datatype, String datatype and Byte Datatype.
 - `public String getDataTypeClassName(int type)`
Given a *type* object we return its class name. In this function we check if our object class is *HOUSE* else throw the *Unknown Datatype Exception*
 - `public int getIdFromClass(Class<?> cls)`
Returns *value id* for *HOUSE* if *cls* is *HOUSE* class.
 - `public TypeInfo getTypeInfoById(int type, long precision, int scale, ExtTypeInfo extTypeInfo)`
Return the *Type Info* ID if the *type* matches the ID for *HOUSE* in *Value* class else throws an *Unknown Datatype Exception*.
 - `public Value getValue(int type, Object data, DataHandler dataHandler)`
If *type* matches *HOUSE type* and *data* is an instance of *HOUSE* we return an object of *HOUSE* datatype.
 - `public Object getObject(Value value, Class<?> cls)`
Converts *Value* object of value class to the specified class, which in our case we convert to a *HOUSE* datatype. If the *cls* is a *HOUSE* datatype and value type is that of *HOUSE* we just create an object of *value* and return it. If not, we convert *value* to *HOUSE* datatype.
 - `public boolean supportsAdd(int type)`
Checks if *type* supports add operation and returns a boolean value.

-
- `public int getAddProofType(int type)`

Checks if *type* has the datatype which can sustain multiple adds without overflow which in our case we check if the datatype is *HOUSE*.

House

The *House* class is used for serialization and to validate the string passed as instance of *House* class. It consists of an *ArrayList* of valid values : *Condominium, Apartment, Townhome, Bungalow, Cottage, Cabin, Chalet, Barndominium, Mansion, Yurt, Castle, Palace, Chateau, Villa, Manor, Fort, Cave* is saved as the variable *houseList*.

- `public boolean equals(Object o)`
Check if the Object passed is equal to the current object of House class.
- `public int hashCode()`
Generates a unique id for the current object which is the index of the Instance variable in the *ArrayList houseList*.
- `public String toString()`
Returns the String format of the given object.
- `public void checkValidity(String house)`
This method checks if the incoming parameter string *house* belongs to the *ArrayList houseList*. If it does not belong to the *houseList* an Exception is thrown by calling the `DbException.get(ErrorCode.INVALID_HOUSE_ERROR)` where `INVALID_HOUSE_ERROR` is a custom error code for the *House Datatype*.

Value

Value is a base class in the h2 database. It provides comparison and conversion methods for all value classes. Since it is a base class, we have only modified it to recognize our *HOUSE* datatype.

- We first initialize the value type for *HOUSE*:
`public static final int HOUSE = 40;`
- Since we have added a new datatype, we increment the total number of value types by one:
`public static final int TYPE_COUNT = HOUSE + 1;`

-
- We then add a case to get the order of the *HOUSE* type:

```
case HOUSE:  
    return 53_000;
```

ValueHouse

valueHouse is an extension of the *Value* class from the package *org.h2.value*. The constructor creates an object of *House* class called *house*

- `public TypeInfo getType()`
Returns the information on the parameters used by the *HOUSE* datatype
- `public StringBuilder getSQL(StringBuilder builder)`
Append the SQL expression for the value *HOUSE* to the string builder
- `public int getValueType()`
Returns the value of the *HOUSE* datatype
- `public String getString()`
Get the value of the *HOUSE* datatype as a string
- `public Object getObject()`
Get the value of the *HOUSE* datatype as an object
- `public int hashCode()`
Get the hash code of the *HOUSE* datatype.
- `public boolean equal(Object other)`
Check whether the two values has the same hash code
- `public int compareTypeSafe(Value o, CompareMode mode)`
Compare the value against another value of the same *HOUSE* datatype
- `public Value convertTo(int targetType, Mode mode, Object column, ExtTypeInfo extTypeInfo)`
Convert a value to the *HOUSE* datatype

TypeInfo

TypeInfo provides information on the parameters used by datatypes available in h2. Since it is already available in the h2 database we only modify it add information on our new *HOUSE* datatype.

-
- We first define *TYPE_HOUSE* as an object of *TypeInfo*
public static final TypeInfo TYPE_HOUSE;
 - We then initialize the typeinfo list with *TYPE_HOUSE*
infos[Value.HOUSE]=TYPE_HOUSE=new
TypeInfo(Value.HOUSE,Integer.MAX_VALUE,0,Integer.MAX_VALUE,null);
 - We then add a case to get the datatype for the value *HOUSE*
case Value.HOUSE:
return TYPE_INFOS_BY_VALUE_TYPE[type];

ErrorCode

ErrorCode defines the codes used for SQL exceptions. Since it is already available in the h2 database we only modify it add the error code for our new *HOUSE* datatype.

```
public static final int INVALID_HOUSE_ERROR = 90147;
```

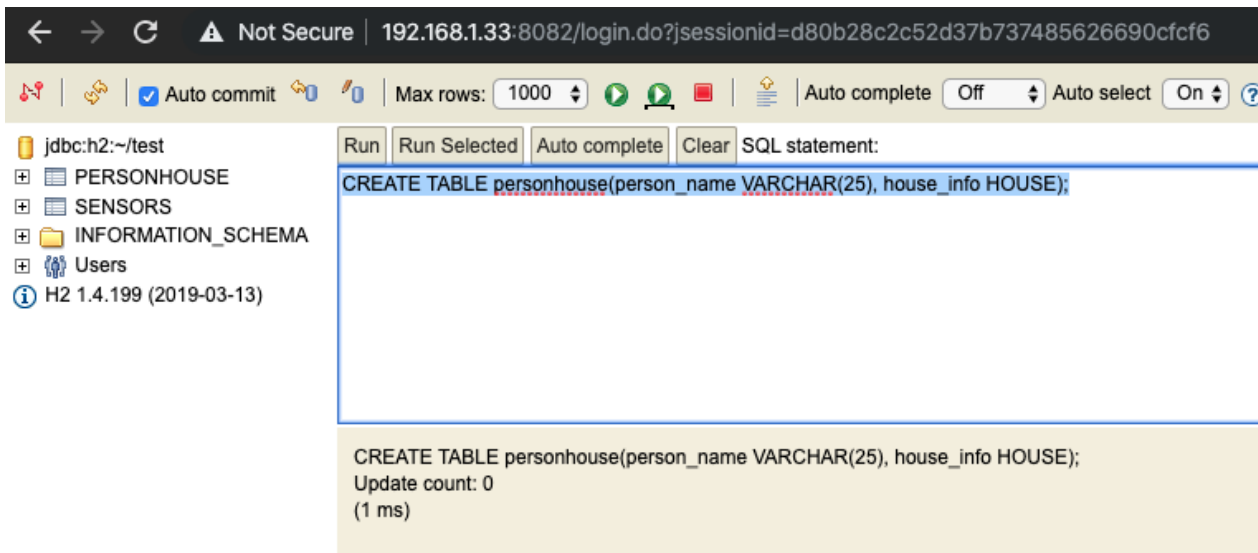
SysProperties

SysProperties contains settings used by the h2 database in order to execute and load classes on a system. It is already available in the h2 database implementation and most of the properties are based on settings used by the machine. However there are some custom properties. To ensure that our *HOUSE* datatype is recognized we add a custom datatype handler property

```
public static final String CUSTOM_DATA_TYPES_HANDLER =  
Utils.getProperty("h2.customDataTypesHandler","org.h2.api.CustomHouseDataType");
```

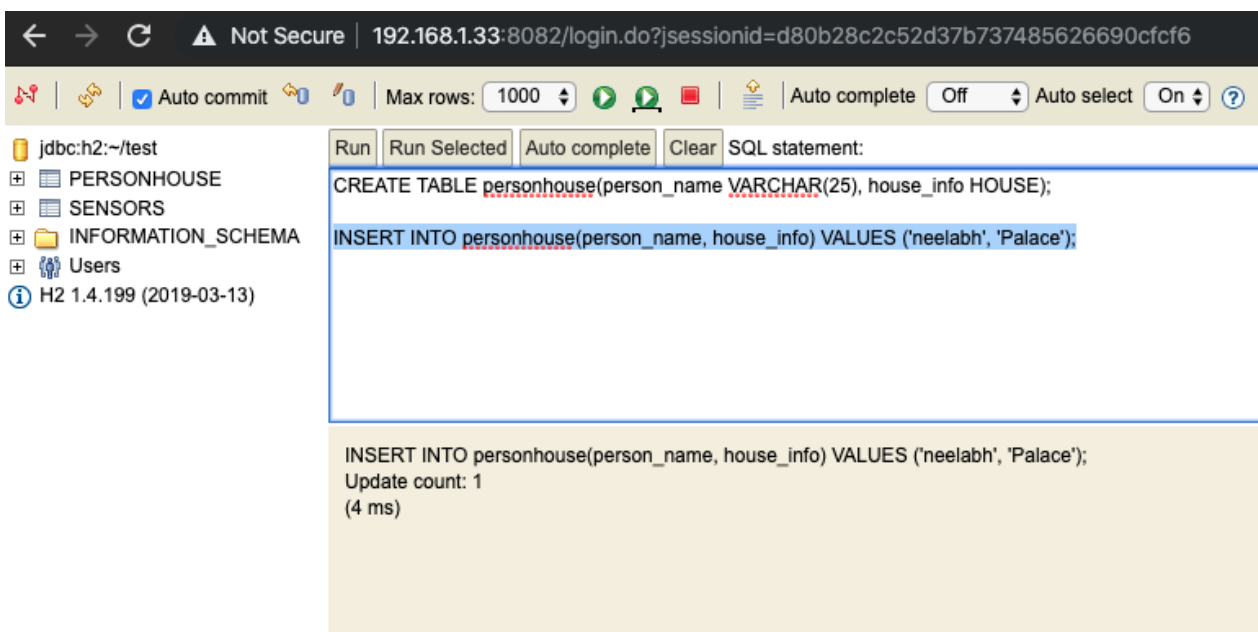
Queries with House Datatype on H2

Create table with house House Datatype



Creating a table

Inserting 1 record into the table



Inserting a record

Show table after inserting 1 record

The screenshot shows a web-based SQL client interface. The top bar displays the URL `192.168.1.33:8082/login.do?jsessionId=d80b28c2c52d37b737485626690cfcf6` and a 'Not Secure' warning. Below the top bar, there are navigation icons and settings like 'Auto commit' (checked), 'Max rows: 1000', 'Auto complete' (Off), and 'Auto select' (On). The left sidebar shows a database tree with 'PERSONHOUSE' selected. The main area displays the SQL statement: `CREATE TABLE personhouse(person_name VARCHAR(25), house_info HOUSE);`, `INSERT INTO personhouse(person_name, house_info) VALUES ('neelabh', 'Palace');`, and `SELECT * FROM personhouse;`. Below the SQL statement, the results of the SELECT query are shown in a table:

PERSON_NAME	HOUSE_INFO
neelabh	Palace

Below the table, it indicates '(1 row, 6 ms)'.

select * from table

Inserting multiple record into the table

The screenshot shows the same web-based SQL client interface. The SQL statement area contains four INSERT statements: `CREATE TABLE personhouse(person_name VARCHAR(25), house_info HOUSE);`, `INSERT INTO personhouse(person_name, house_info) VALUES ('neelabh', 'Palace');`, `INSERT INTO personhouse(person_name, house_info) VALUES ('Kavya', 'Bungalow');`, `INSERT INTO personhouse(person_name, house_info) VALUES ('Prashanth', 'Cave');`, and `INSERT INTO personhouse(person_name, house_info) VALUES ('Siddarth', 'Cabin');`. Below the SQL statement, the results of the INSERT statements are shown in a table:

PERSON_NAME	HOUSE_INFO
neelabh	Palace
Kavya	Bungalow
Prashanth	Cave
Siddarth	Cabin

Below the table, it indicates '(1 row, 6 ms)'.

Inserting multiple records

```
501 35306 55650 0 4:09PM ttys000 0:00.00 grep --color=auto --exclude-dir=.bzip --exclude-dir=CVS --exclude-dir=.git --exclude-dir=.hg --exclude-dir=.svn h2
neelabhkumar@neelabhs-MBP ~/Documents/h2db/h2/bin ./h2.sh
1340
coming inside
coming inside string
[Condominium, Apartment, Townhome, Bungalow, Cottage, Cabin, Chalet, Barndominium, Mansion, Yurt, Castle, Palace, Chateau, Villa, Manor, Fort, Cave]
Palace
a new object created
1340
coming inside
coming inside string
[Condominium, Apartment, Townhome, Bungalow, Cottage, Cabin, Chalet, Barndominium, Mansion, Yurt, Castle, Palace, Chateau, Villa, Manor, Fort, Cave]
Bungalow
a new object created
1340
coming inside
coming inside string
[Condominium, Apartment, Townhome, Bungalow, Cottage, Cabin, Chalet, Barndominium, Mansion, Yurt, Castle, Palace, Chateau, Villa, Manor, Fort, Cave]
Cave
a new object created
1340
coming inside
coming inside string
[Condominium, Apartment, Townhome, Bungalow, Cottage, Cabin, Chalet, Barndominium, Mansion, Yurt, Castle, Palace, Chateau, Villa, Manor, Fort, Cave]
Cabin
a new object created
```

stack Output for the insertion

Show table after inserting multiple record

The screenshot shows the H2 database console interface. On the left, the database structure is visible, including 'PERSONHOUSE', 'SENSORS', 'INFORMATION_SCHEMA', and 'Users'. The main area displays three SQL insert statements for the 'personhouse' table, followed by a 'SELECT * FROM personhouse;' query. The query result is shown as a table with 4 rows and 2 columns: 'PERSON_NAME' and 'HOUSE_INFO'.

Run Run Selected Auto complete Clear SQL statement:

```
INSERT INTO personhouse(person_name, house_info) VALUES ('Kavya', 'Bungalow');
INSERT INTO personhouse(person_name, house_info) VALUES ('Prashanth', 'Cave');
INSERT INTO personhouse(person_name, house_info) VALUES ('Siddarth', 'Cabin');

SELECT * FROM personhouse;
```

PERSON_NAME	HOUSE_INFO
neelabh	Palace
Kavya	Bungalow
Prashanth	Cave
Siddarth	Cabin

(4 rows, 3 ms)

select * from table

Exception thrown after entering a record with invalid House datatype

```
jdbc:h2:~/test
PERSONHOUSE
SENSORS
INFORMATION_SCHEMA
Users
H2 1.4.199 (2019-03-13)

Run Run Selected Auto complete Clear SQL statement:

INSERT INTO personhouse(person_name, house_info) VALUES ('Prashanth', 'Cave');
INSERT INTO personhouse(person_name, house_info) VALUES ('Siddarth', 'Cabin');
INSERT INTO personhouse(person_name, house_info) VALUES ('ratul', 'Tajmahal');

SELECT * FROM personhouse;

INSERT INTO personhouse(person_name, house_info) VALUES ('ratul', 'Tajmahal');

(Message 90147 not found); SQL statement:
INSERT INTO personhouse(person_name, house_info) VALUES ('ratul', 'Tajmahal') -- ('ratul', 'Tajmahal') [90147-199] 90147/90147 (Help)
org.h2.jdbc.JdbcSQLException: (Message 90147 not found); SQL statement:
INSERT INTO personhouse(person_name, house_info) VALUES ('ratul', 'Tajmahal') -- ('ratul', 'Tajmahal') [90147-199]
at org.h2.message.DbException.getJdbcSQLException(DbException.java:624)
at org.h2.message.DbException.getJdbcSQLException(DbException.java:427)
at org.h2.message.DbException.get(DbException.java:205)
at org.h2.message.DbException.get(DbException.java:181)
at org.h2.message.DbException.get(DbException.java:170)
at org.h2.value.Value.checkValidity(Value.java:65)
at org.h2.value.Value.<init>(Value.java:25)
at org.h2.api.CustomHouseDataType.convert(CustomHouseDataType.java:65)
at org.h2.value.Value.convertTo(Value.java:828)
at org.h2.value.Value.convertTo(Value.java:738)
at org.h2.table.Column.convert(Column.java:199)
at org.h2.command.dml.Insert.insertRows(Insert.java:164)
at org.h2.command.dml.Insert.update(Insert.java:132)
at org.h2.command.CommandContainer.update(CommandContainer.java:133)
at org.h2.command.Command.executeUpdate(Command.java:267)
at org.h2.jdbc.JdbcStatement.executeInternal(JdbcStatement.java:233)
at org.h2.jdbc.JdbcStatement.execute(JdbcStatement.java:1113)
at org.h2.server.web.WebApp.getResult(WebApp.java:1436)
at org.h2.server.web.WebApp.query(WebApp.java:1111)
at org.h2.server.web.WebApp$1.next(WebApp.java:1073)
at org.h2.server.web.WebApp$1.next(WebApp.java:1060)
at org.h2.server.web.WebThread.process(WebThread.java:173)
at org.h2.server.web.WebThread.run(WebThread.java:93)
at java.base/java.lang.Thread.run(Thread.java:844)
```

Invalid insertion of HOUSE as TajMahal

```
coming inside
coming inside string
[Condominium, Apartment, Townhome, Bungalow, Cottage, Cabin, Chalet, Barndominium, Mansion, Yurt, Castle, Palace, Chateau, Villa, Manor, Fort, Cave]
Tajmahal
Invalid House Type : Throwing an exception
```

stack output on wrong insertion

Display records that matches the *equality* condition

The screenshot shows a database client interface with a sidebar on the left containing a tree view of the database schema. The main area displays SQL queries and their results. The first query is `SELECT * FROM PERSONHOUSE;`, which returns 4 rows. The second query is `SELECT PERSON_NAME FROM PERSONHOUSE WHERE HOUSE_INFO='Palace';`, which returns 1 row.

SQL statement:

```
SELECT * FROM PERSONHOUSE;
```

PERSON_NAME	HOUSE_INFO
neelabh	Palace
Kavya	Bungalow
Prashanth	Cave
Siddarth	Cabin

(4 rows, 1 ms)

```
SELECT PERSON_NAME FROM PERSONHOUSE WHERE HOUSE_INFO='Palace';
```

PERSON_NAME
neelabh

(1 row, 0 ms)

Equality search









Information Schema

The screenshot shows a database client interface with a sidebar on the left containing a tree view of the database schema. The main area displays the SQL statement `SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_NAME='PERSONHOUSE';` and its results. The results table shows the table 'PERSONHOUSE' in the 'PUBLIC' schema, with a row count of 4.

TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE	STORAGE_TYPE	REMARKS	LAST_MODIFICATION	ID	TYPE_NAME	TABLE_CLASS	ROW_COUNT_ESTIMATE
TEST	PUBLIC	PERSONHOUSE	TABLE	CACHED	CREATE CACHED TABLE "PUBLIC"."PERSONHOUSE" ("PERSON_NAME" VARCHAR(25), "HOUSE_INFO" HOUSE)	0	7	null	org.h2.mvstore.db.MVTable	4

(1 row, 2 ms)

Information Schema showing the type HOUSE

		PERSONHOUSE
		PERSON_NAME
		VARCHAR(25)
		HOUSE_INFO
		HOUSE

PERSONHOUSE table showing the type HOUSE