IntegratedML Syntax, Beta Version 0.3 (build 243)

Table of Contents

Model Creation

Defines the "problem" for which to train models, allowing explicit naming / typing / ordering for PREDICTING and WITH values:

```
CREATE MODEL model-name PREDICTING (value-name1 datatype1)
   WITH (value-name3 datatype3, value-name4 datatype4, ...)
   [ USING <alg description> ]
```

A FROM clause allows implied typing / ordering for PREDICTING and WITH columns/values. The FROM clause itself is NOT executed during CREATE MODEL (beyond an info compile to acquire the columns and types), and no training takes place. Instead, the clause is recorded as for a non-materialized view, and can be used implicitly by TRAIN MODEL.

Important detail: The implied WITH value list is the result of the FROM as if it were SELECT * FROM ... with the value named (via its column alias) in the PREDICTING clause removed and remaining columns in their selected order. This semantics is used consistently whenever a FROM clause is used to implicitly produce PREDICTING and WITH columns/values.

```
CREATE MODEL model-name PREDICTING (value-name1)
   FROM table-a [JOIN view-b ON ...] [WHERE field1 = ...]
   [ USING <alg description>  ]
CREATE MODEL model-name PREDICTING (value-name1)
   FROM (SELECT with-expr1, with-expr2, ... FROM table-a LEFT OUTER JOIN
view-b ... WHERE ...)
   [ USING <alg description>  ]
```

Maximally simplified:

```
CREATE MODEL model-name PREDICTING (column-name) FROM table-name
```

Proposed Backus-Naur Form (BNF)

```
CREATE MODEL <model-name> PREDICTING ( <ident> [ <datyp> ] ) [ <with-clause>
] [ <model-source> ] [ <using-clause> ]
     <model-name> ::= <ident> -- must be unique in the set of %ML.Model.Name
values.
     <model-column> ::= <ident> <datyp>
          <ident> ::= <identifier> | <delimited identifier> - see sqlbnf for
more info on <identifier> and <delimited identifier>
          <datyp> ::= <datyp spec> - see sqlbnf for more info
     <with-clause> ::= WITH ( <model-column> { , <model-column> } )
     <model-source> ::= FROM <model-source-query>
          <model-source query> ::= ( <selex> ) |
                                   <from> [ WHERE <prdct> ] [ <group> ] [ <having>
] ] [ <order> ] - see sqlbnf for more info on <selex>, <from>, <prdct>,
<group>, <having>, and <order>
     <using-clause> ::= USING <alg options>
          <alg options> ::= <json-object-string>
               <json-object-string> ::= "{" { <literal> : <literal> } "}" -
option "ml_provider":<value> will define %ML.Model.DefautProvider with
<value>. All other options in the JSON string are recorded in the
%ML.Model.DefaultParameters array collection.
```

Examples:

Statement:

```
create model sample.mdl predicting ( FavoriteColors varchar(30)) from (select
name, ssn, favoritecolors from sample.person)
```

Statement:

```
create model sample.md4 predicting ( favoritecolors ) from sample.person
where id > 0 order by name
```

Model Maintenance

Delete Models

```
DROP MODEL model-name.
```

Execution of DROP MODEL will delete the row/object in the %ML.Model class where %ML.Model.Name = <model-name>, plus associated TrainingRun, TrainedModel, ValidationRun and ValidationStats records.

Proposed BNF

```
DROP MODEL <model-name>
<model-name> ::= <ident>
```

Alter models

A limited number of maintenance operations are allowed on models:

```
ALTER MODEL model-name PURGE
```

This command will drop all TrainingRun, TrainedModel, ValidationRun and ValidationStats records associated with this model, except for those rows associated with the current DefaultTrainingModel.

```
ALTER MODEL model-name DEFAULT trained-model-name
```

This command will change the model's DefaultTrainedModel reference to point to the named TrainedModel record, or throw an error if no such TrainedModel is found.

Proposed BNF

ALTER MODEL <model-name> <alter-action>
<alter-action> ::= PURGE [<scope>]
| <DefaultTrainedModel>
<DefaultTrainedModel> ::= DEFAULT [TRAINED MODEL] <trained-model-name>
<scope> ::= ALL | <integer> DAYS
<scope> defaults to ALL

Other operations

Any direct DML (INSERT, UPDATE, DELETE) against the %ML tables from outside of our

own code should preferably throw an error to avoid confusion and corruption.

Model Training

Either field expressions can be assigned to values in FOR and/or WITH explicitly, via a list matching the CREATE MODEL ordering, or the same implied PREDICTING/WITH value list construction described for CREATE MODEL ... FROM can be used. If no FROM clause is provided, the FROM clause from CREATE MODEL is re-used (if no FROM clause was specified in CREATE MODEL the FROM clause for TRAIN MODEL is mandatory).

Training a model will immediately create a %ML.TrainingRun instance and kick off training (in the background). The TrainingRun record can be consulted (using simple SELECT syntax) to check on training progress. This TrainingRun has its name initialized with the %ML.Model name, concatenated with a running integer if a TrainingRun with that name for that model already exists. This default name can be overridden through an AS clause, which will throw an error if a TrainingRun record with the specified preferred name already exists. Handling naming at the TrainingRun level allows us to do the checking at the time TRAIN MODEL is called, because TrainedModel records may only be created much later.

When training concludes, one (or possibly more, for some providers) %ML.TrainedModel records will have been created for this TrainingRun. These will take the name of the TrainingRun they are created for, optionally with a running integer to avoid duplicate names (per model). The provider will set the model's DefaultTrainedModel reference to one of these TrainedModel records (provider's discretion, in 99% of cases there'll be a single TrainedModel anyhow), except if the NOT DEFAULT keyword was used in the TRAIN MODEL command.

```
TRAIN MODEL model-name
   [AS preferred-name ]
   [NOT DEFAULT ]
   [FOR field-expr1 ]
   [WITH value-name3 = field-expr3, value-name4 = field-expr4, ... | WITH
(field-expr3, field-expr4,...)]
   [FROM table-a [JOIN table-b ON ...] [WHERE field1 = ...]]
   [ USING <alg options>  ]
TRAIN MODEL model-name
   [AS preferred-name ]
   [NOT DEFAULT ]
   [FOR field-expr1 ]
   [WITH value-name3 = field-expr3, value-name4 = field-expr4, ... | WITH
(field-expr3, field-expr4,...)]
   [FROM (SELECT with-expr1, with-expr2, ..., value-name1, ..., with-expr3,
... FROM table1 LEFT OUTER JOIN table2 ... WHERE ...)]
   [ USING <alg options> ]
```

Maximally simplified:

```
TRAIN MODEL model-name
```

Proposed BNF

TRAIN MODEL <model-name> [AS <model-name>] [NOT DEFAULT] [FOR <model-column>] [<with-clause-train>] [<model-source>] [<using-clause>]

<with-clause-train> ::= WITH <train-with>

<train-with> ::= (<model-column> { , <model-column> })
| <with-value-assignment>
<with-value-assignment> ::= <ident> = <expresion> { , <ident> = <expresion> }

See CREATE MODEL for more details

Model Execution

Prediction

```
PREDICT ( <model-name> [ USE <trained-model-name> ] [ <predict-with> ] )
<predict-with> ::= <with-names> | <with-list>
<with-names> ::= WITH <field-name> = <expression> { , <field-name> =
<expression> }
<with-list> ::= WITH ( [ <expression> ] { , [ <expression> ] } )
<model-name> ::= <ident>
<expression> ::= <scalx>
```

The PREDICT() function takes a model name and an optional WITH clause. PREDICT applies the specified trained model to predict the result for each row in the applicable row-set, using the row input values as specified by the WITH clause. If the WITH clause is missing then, by default, fields from the applicable FROM-clause are used, using the exact same field names as was specified in CREATE MODEL (these field names must be unambiguously found in the FROM tables). If <with-names> is used, the names must be from CREATE MODEL, their order does not matter, and any missing field names are taken from the FROM clause as above. If <with-list> is used, the arguments are ordered and numbered exactly as specified in CREATE MODEL, with missing arguments (either between commas or trailing) taken by name from the FROM clause as above. USE may specify a named trained model to use, otherwise the default trained model is used.

Maximally simplified:

```
SELECT PREDICT(<model-name>), * FROM <table-name>
```

Probability

```
PROBABILITY ( <model-name> [ USE <trained-model-name> ] [ FOR <expression> ]
[ <predict-with> ] )
```

The syntax is identical to PREDICT, except for the added FOR clause. The WITH clause is identical to PREDICT. PROBABILITY applies the specified trained model to compute, for each

row in the applicable row-set, the probability (from 0 to 1) that the value specified by FOR is actually the correct result for the model. The model must be a "categorization"-type model, PROBABILITY cannot be used for a "regression"-type model. When FOR is not specified, the value 1 is used, but only for "binary"-type models, i.e. models whose result can be only 0 or 1, otherwise FOR must be specified.

[The above FOR default doesn't make much sense. Maybe if FOR is omitted, it should return the probability of the PREDICT() result?]

Maximally simplified:

```
SELECT PROBABILITY(<model-name> FOR 42), * FROM <table-name>
```

Model Validation (! Future Feature !)

```
VALIDATE MODEL model-name
    [ USE trained-model-name]
    [ WITH value-name3 = field-expr3, value-name4 = field-expr4, ... ]
    FROM query
    [ USING <validation options> ]
```

Model validation, like training, may take time and therefore a similar ValidationRun record is created immediately upon calling VALIDATE MODEL, allowing progress to be tracked by simply querying the %ML.ValidationRun table. The USING clause is a placeholder for future use, for example to limit what is being measured (only overall precision or per-category, micro vs macro averages, ...).

- %ML.ValidationRun
  - TrainedModelID (FK)
  - StartedAt
  - CompletedAt
  - Status
  - ValidationQuery (SQL statement used for VALIDATE MODEL or TRAIN MODEL)
  - ValidationParameters (USING clause payload)
  - TrainingRunID (FK, only used when a byproduct of TRAIN MODEL)
- %ML.ValidationStats
  - ValidationRunID (FK)
  - MetricName (such as precision, recall, ...)
  - MetricValue
  - TargetValue (for per-category metrics, null otherwise)

Proposed BNF

```
VALIDATE MODEL <model-name>
    [ USE <model-name> ]   -- trained model name
    [ WITH value-name3 = field-expr3, value-name4 = field-expr4, ... ]
```

```
FROM <model-source-query>
[ <using-clause> ]
```