public class |

Model

A Model represents a table in the database. Instances of this class represent a database row.

Model instances operate with the concept of a `dataValues` property, which stores the actual values represented by the instance. By default, the values from dataValues can also be accessed directly from the Instance, that is:

```
instance.field
// is the same as
instance.get('field')
// is the same as
instance.getDataValue('field')
```

However, if getters and/or setters are defined for `field` they will be invoked, instead of returning the value from `dataValues`. Accessing properties directly or using `get` is preferred for regular use, `getDataValue` should only be used for custom getters.

See:

Sequelize#define for more information about getters and setters

Static Method Summary

Static Public Methods	
public static	addScope(name: String, scope: Object | Function, options: Object) Add a new scope to the model.
public static	aggregate(field: String, aggregateFunction: String, options: Object): Promise<DataTypes|object> Run an aggregation method on the specified field
public static	belongsTo(target: Model, options: object): BelongsTo Creates an association between this (the source) and the provided target.
public static	belongsToMany(target: Model, options: object): BelongsToMany Create an N:M association with a join table.
public static	build(options: Object): Model | Model[] Builds a new model instance.
public static	bulkCreate(records: Array, options: Object): Promise<Array<Model>> Create and insert multiple instances in bulk.
public static	count(options: Object): Promise<Integer> Count the number of records matching the provided where clause.
public static	create(values: Object, options: Object): Promise<Model> Builds a new model instance and calls save on it.
public static	decrement(fields: *, options: *): Promise<this> since 4.36.0 Decrement the value of one or more columns.
public static	describe(schema: *, options: *): Promise Run a describe query on the table.
public static	destroy(options: Object): Promise<Integer> Delete multiple instances, or set their deletedAt timestamp to the current time if `paranoid` is enabled.
public static	drop(options: Object): Promise Drop the table represented by this Model
public static	findAll(options: Object): Promise<Array<Model>> Search for multiple instances.

public static	findAndCount(findOptions: Object): Promise<{count: Integer, rows: Model[]}> Find all the rows matching your query, within a specified offset / limit, and get the total number of rows matching your query.
public static	findById(id: Number | String | Buffer, options: Object): Promise<Model> Search for a single instance by its primary key.
public static	findCreateFind(options: Object): Promise<Model, created> A more performant findOrCreate that will not work under a transaction (at least not in postgres) Will execute a find call, if empty then attempt to create, if unique constraint then attempt to find again
public static	findOne(options: Object): Promise<Model> Search for a single instance.
public static	findOrBuild(options: Object): Promise<Model, initialized> Find a row that matches the query, or build (but don't save) the row if none is found. The successful result of the promise will be (instance, initialized) - Make sure to use .spread() **Alias**: *findOrInitialize*
public static	findOrCreate(options: Object): Promise<Model, created> Find a row that matches the query, or build and save the row if none is found The successful result of the promise will be (instance, created) - Make sure to use .spread()
public static	getTableName(): String | Object Get the tablename of the model, taking schema into account.
public static	hasMany(target: Model, options: object): HasMany Creates a 1:m association between this (the source) and the provided target.
public static	hasOne(target: Model, options: object): HasOne Creates an association between this (the source) and the provided target.
public static	increment(fields: String | Array | Object, options: Object): Promise<this> Increment the value of one or more columns.
public static	init(attributes: Object, options: Object): Model Initialize a model, representing a table in the DB, with attributes and options.
public static	max(field: String, options: Object): Promise<Any> Find the maximum value of field
public static	min(field: String, options: Object): Promise<Any> Find the minimum value of field
public static	removeAttribute(attribute: String) Remove attribute from model definition
public static	restore(options: Object): Promise<undefined> Restore multiple instances if `paranoid` is enabled.
public static	schema(schema: String, options: Object): this Apply a schema to this model.
public static	scope(options: Array | Object | String | null): Model Apply a scope created in `define` to the model.
public static	sum(field: String, options: Object): Promise<Number> Find the sum of field
public static	sync(options: *): Promise<this> Sync this Model to the DB, that is create the table.

public static	truncate(options: object): Promise Truncate all instances of the model.
public static	unscoped(): Model
public static	update(values: Object, options: Object): Promise<Array<affectedCount, affectedRows>> Update multiple instances that match the where options.
public static	upsert(values: Object, options: Object): Promise<created> Insert or update a single row.

Constructor Summary

Public Constructor	
public	constructor(values: Object, options: Object) Builds a new model instance.

Member Summary

Public Members	
public	isNewRecord: Boolean: * Returns true if this instance has not yet been persisted to the database
public get	sequelize: Sequelize: * A reference to the sequelize instance

Method Summary

Public Methods		
public	changed(key: String): Boolean | Array If changed is called with a string it will return a boolean indicating whether the value of that key in `dataValues` is different from the value in `_previousDataValues`.	
public	decrement(fields: String | Array | Object, options: Object): Promise Decrement the value of one or more columns.	
public	destroy(options: Object): Promise<undefined> Destroy the row corresponding to this instance.	
public	equals(other: Model): Boolean Check whether this and `other` Instance refer to the same row	
public	equalsOneOf(others: Array): Boolean Check if this is equal to one of `others` by calling equals	
public	get(key: String, options: Object): Object | any If no key is given, returns all values of the instance, also invoking virtual getters.	
public	getDataValue(key: String): any Get the value of the underlying data value	
public	increment(fields: String | Array | Object, options: Object): Promise<this> Increment the value of one or more columns.	since 4.0.0
public	isSoftDeleted(): Boolean Helper method to determine if a instance is "soft deleted".	
public	previous(key: String): any | Array<any> Returns the previous value for key from `_previousDataValues`.	
public	reload(options: Object): Promise<this> Refresh the current instance in-place, i.e.	

public	restore(options: Object): Promise<undefined> Restore the row corresponding to this instance.
public	save(options: Object): Promise<this|Errors.ValidationError> Validate this instance, and if the validation passes, persist it to the database.
public	set(key: String | Object, value: any, options: Object): * Set is used to update values on the instance (the sequelize representation of the instance that is, remember that nothing will be persisted before you actually call `save`).
public	setDataValue(key: String, value: any) Update the underlying data value
public	toJSON(): object Convert the instance to a JSON representation.
public	update(updates: Object, options: Object): Promise<this> This is the same as calling `set` and then calling `save` but it only saves the exact values passed to it, making it more atomic and safer.
public	validate(options: Object): Promise<undefined> Validate the attributes of this instance according to validation rules set in the model definition.
public	where(checkVersion: *): Object Get an object representing the query for this instance, use with `options.where`

Static Public Methods

public static addScope(name: String, scope: Object | Function, options: Object)

Add a new scope to the model. This is especially useful for adding scopes with includes, when the model you want to include is not available at the time this model is defined.

By default this will throw an error if a scope with that name already exists. Pass `override: true` in the options object to silence this error.

Params:

Name	Type	Attribute	Description
name	String		The name of the scope. Use `defaultScope` to override the default scope
scope	Object | Function		
options	Object	optional	
options.override	Boolean	optional default: false	

public static aggregate(field: String, aggregateFunction: String, options: Object): Promise<DataTypes|object>

Run an aggregation method on the specified field

Params:

Name	Type	Attribute	Description
field	String		The field to aggregate over. Can be a field name or *
aggregateFunction	String		The function to use for aggregation, e.g. sum, max etc.
options	Object	optional	Query options. See sequelize.query for full options

options.where	Object	optional	A hash of search attributes.
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).
options.dataType	DataTypes | String	optional	The type of the result. If `field` is a field in this Model, the default will be the type of that field, otherwise defaults to float.
options.distinct	boolean	optional	Applies DISTINCT to the field being aggregated over
options.transaction	Transaction	optional	Transaction to run query under
options.plain	Boolean	optional	When `true` , the first returned value of `aggregateFunction` is cast to `dataType` and returned. If additional attributes are specified, along with `group` clauses, set `plain` to `false` to return all values of all returned rows. Defaults to `true`

Return:

Promise<DataTypes|object>	Returns the aggregate result cast to `options.dataType` , unless `options.plain` is false, in which case the complete data result is returned.

public static belongsTo(target: Model, options: object): BelongsTo

Creates an association between this (the source) and the provided target. The foreign key is added on the source.

Params:

Name	Type	Attribute	Description
target	Model		
options	object	optional	
options.hooks	boolean	optional default: false	Set to true to run before-/afterDestroy hooks when an associated model is deleted because of a cascade. For example if `User.hasOne(Profile, {onDelete: 'cascade', hooks:true})` , the before-/afterDestroy hooks for profile will be called when a user is deleted. Otherwise the profile will be deleted without invoking any hooks
options.as	string	optional	The alias of this model, in singular form. See also the `name` option passed to `sequelize.define` . If you create multiple associations between the same tables, you should provide an alias to be able to distinguish between them. If you provide an alias when creating the association, you should provide the same alias when eager loading and when getting associated models. Defaults to the singularized name of target
options.foreignKey	string | object	optional	The name of the foreign key in the source table or an object representing the type definition for the foreign column (see `Sequelize.define` for syntax). When using an object, you can add a `name` property to set the name of the column. Defaults to the name of target + primary key of target

options.targetKey	string	optional	The name of the field to use as the key for the association in the target table. Defaults to the primary key of the target table
options.onDelete	string	optional default: 'SET NULL|NO ACTION'	SET NULL if foreignKey allows nulls, NO ACTION if otherwise
options.onUpdate	string	optional default: 'CASCADE'	
options.constraints	boolean	optional default: true	Should on update and on delete constraints be enabled on the foreign key.

Return:

BelongsTo

Example:

```
Profile.belongsTo(User) // This will add userId to the profile table
```

public static belongsToMany(target: Model, options: object): BelongsToMany

Create an N:M association with a join table. Defining `through` is required.

Params:

Name	Type	Attribute	Description
target	Model		
options	object		
options.hooks	boolean	optional default: false	Set to true to run before-/afterDestroy hooks when an associated model is deleted because of a cascade. For example if `User.hasOne(Profile, {onDelete: 'cascade', hooks:true})` , the before-/afterDestroy hooks for profile will be called when a user is deleted. Otherwise the profile will be deleted without invoking any hooks
options.through	Model | string | object		The name of the table that is used to join source and target in n:m associations. Can also be a sequelize model if you want to define the junction table yourself and add extra attributes to it.
options.through.model	Model	optional	The model used to join both sides of the N:M association.
options.through.scope	object	optional	A key/value set that will be used for association create and find defaults on the through model. (Remember to add the attributes to the through model)
options.through.unique	boolean	optional default: true	If true a unique key will be generated from the foreign keys used (might want to turn this off and create specific unique keys when using scopes)

options.as	string | object	optional	The alias of this association. If you provide a string, it should be plural, and will be singularized using node.inflection. If you want to control the singular version yourself, provide an object with `plural` and `singular` keys. See also the `name` option passed to `sequelize.define`. If you create multiple associations between the same tables, you should provide an alias to be able to distinguish between them. If you provide an alias when creating the association, you should provide the same alias when eager loading and when getting associated models. Defaults to the pluralized name of target
options.foreignKey	string | object	optional	The name of the foreign key in the join table (representing the source model) or an object representing the type definition for the foreign column (see `Sequelize.define` for syntax). When using an object, you can add a `name` property to set the name of the column. Defaults to the name of source + primary key of source
options.otherKey	string | object	optional	The name of the foreign key in the join table (representing the target model) or an object representing the type definition for the other column (see `Sequelize.define` for syntax). When using an object, you can add a `name` property to set the name of the column. Defaults to the name of target + primary key of target
options.scope	object	optional	A key/value set that will be used for association create and find defaults on the target. (sqlite not supported for N:M)
options.timestamps	boolean	optional default: sequelize.options.timestamps	Should the join model have timestamps
options.onDelete	string	optional default: 'SET NULL|CASCADE'	Cascade if this is a n:m, and set null if it is a 1:m
options.onUpdate	string	optional default: 'CASCADE'	
options.constraints	boolean	optional default: true	Should on update and on delete constraints be enabled on the foreign key.

Return:

BelongsToMany

Example:

```
// Automagically generated join model
User.belongsToMany(Project, { through: 'UserProjects' })
Project.belongsToMany(User, { through: 'UserProjects' })

// Join model with additional attributes
```

```
})
User.belongsToMany(Project, { through: UserProjects })
Project.belongsToMany(User, { through: UserProjects })
```

public static build(options: Object): Model | Model[]

Builds a new model instance.

Params:

Name	Type	Attribute	Description
(values|values[])	Object	optional default: {}	An object of key value pairs or an array of such. If an array, the function will return an array of instances.
options	Object	optional	
options.raw	Boolean	optional default: false	If set to true, values will ignore field and virtual setters.
options.isNewRecord	Boolean	optional default: true	
options.include	Array	optional	an array of include options - Used to build prefetched/included model instances. See set

Return:

Model | Model[]

public static bulkCreate(records: Array, options: Object): Promise<Array<Model>>

Create and insert multiple instances in bulk.

The success handler is passed an array of instances, but please notice that these may not completely represent the state of the rows in the DB. This is because MySQL and SQLite do not make it easy to obtain back automatically generated IDs and other default values in a way that can be mapped to multiple records. To obtain Instances for the newly created values, you will need to query for them again.

If validation fails, the promise is rejected with an array-like AggregateError

Params:

Name	Type	Attribute	Description
records	Array		List of objects (key/value pairs) to create instances from
options	Object	optional	
options.fields	Array	optional	Fields to insert (defaults to all fields)
options.validate	Boolean	optional default: false	Should each row be subject to validation before it is inserted. The whole insert will fail if one row fails validation
options.hooks	Boolean	optional default: true	Run before / after bulk create hooks?
options.individualHooks	Boolean	optional default: false	Run before / after create hooks for each individual Instance? BulkCreate hooks will still be run if options.hooks is true.

options.ignoreDuplicates	Boolean	optional default: false	Ignore duplicate values for primary keys? (not supported by postgres)
options.updateOnDuplicate	Array	optional	Fields to update if row key already exists (on duplicate key update)? (only supported by mysql). By default, all fields are updated.
options.transaction	Transaction	optional	Transaction to run query under
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).
options.returning	Boolean	optional default: false	Append RETURNING * to get back auto generated values (Postgres only)
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)

Return:
Promise<Array<Model>>

public static count(options: Object): Promise<Integer>

Count the number of records matching the provided where clause.

If you provide an `include` option, the number of matching associations will be counted instead.

Params:

Name	Type	Attribute	Description
options	Object	optional	
options.where	Object	optional	A hash of search attributes.
options.include	Object	optional	Include options. See `find` for details
options.paranoid	Boolean	optional default: true	Set `true` to count only non-deleted records. Can be used on models with `paranoid` enabled
options.distinct	Boolean	optional	Apply COUNT(DISTINCT(col)) on primary key or on options.col.
options.col	String	optional	Column on which COUNT() should be applied
options.attributes	Object	optional	Used in conjunction with `group`
options.group	Object	optional	For creating complex counts. Will return multiple rows as needed.
options.transaction	Transaction	optional	Transaction to run query under
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).

options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)

Return:
Promise<Integer>

public static create(values: Object, options: Object): Promise<Model>

Builds a new model instance and calls save on it.

Params:

Name	Type	Attribute	Description
values	Object		
options	Object	optional	
options.raw	Boolean	optional default: false	If set to true, values will ignore field and virtual setters.
options.isNewRecord	Boolean	optional default: true	
options.include	Array	optional	an array of include options - Used to build prefetched/included model instances. See set
options.fields	Array	optional	If set, only columns matching those in fields will be saved
options.fields	string[]	optional	An optional array of strings, representing database columns. If fields is provided, only those columns will be validated and saved.
options.silent	Boolean	optional default: false	If true, the updatedAt timestamp will not be updated.
options.validate	Boolean	optional default: true	If false, validations won't be run.
options.hooks	Boolean	optional default: true	Run before and after create / update + validate hooks
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).
options.transaction	Transaction	optional	Transaction to run query under
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)
options.returning	Boolean	optional default: true	Return the affected rows (only for postgres)

Return:
Promise<Model>

Model#build
Model#save

public static decrement(fields: *, options: *): Promise<this>

since 4.36.0

Decrement the value of one or more columns. This is done in the database, which means it does not use the values currently stored on the Instance. The decrement is done using a `sql SET column = column - X WHERE foo = 'bar'` query. To get the correct value after a decrement into the Instance you should do a reload.

```
// decrement number by 1
Model.decrement('number', { where: { foo: 'bar' });

// decrement number and count by 2
Model.decrement(['number', 'count'], { by: 2, where: { foo: 'bar' } });

// decrement answer by 42, and decrement tries by -1.
// `by` is ignored, since each column has its own value
Model.decrement({ answer: 42, tries: -1}, { by: 2, where: { foo: 'bar' } });
```

Params:

Name	Type	Attribute	Description
fields	*		
options	*		

Return:
Promise<this>

See:
Model#increment
Model#reload

public static describe(schema: *, options: *): Promise

Run a describe query on the table. The result will be return to the listener as a hash of attributes and their types.

Params:

Name	Type	Attribute	Description
schema	*		
options	*		

Return:
Promise

public static destroy(options: Object): Promise<Integer>

Delete multiple instances, or set their deletedAt timestamp to the current time if `paranoid` is enabled.

Params:

Name	Type	Attribute	Description
options	Object		
options.where	Object	optional	Filter the destroy
options.hooks	Boolean	optional default: true	Run before / after bulk destroy hooks?
options.individualHooks	Boolean	optional default: false	If set to true, destroy will SELECT all records matching the where parameter and will execute before / after destroy hooks on each row
options.limit	Number	optional	How many rows to delete

options.force	Boolean	optional default: false	Delete instead of setting deletedAt to current timestamp (only applicable if `paranoid` is enabled)
options.truncate	Boolean	optional default: false	If set to true, dialects that support it will use TRUNCATE instead of DELETE FROM. If a table is truncated the where and limit options are ignored
options.cascade	Boolean	optional default: false	Only used in conjunction with TRUNCATE. Truncates all tables that have foreign-key references to the named table, or to any tables added to the group due to CASCADE.
options.restartIdentity	Boolean	optional default: false	Only used in conjunction with TRUNCATE. Automatically restart sequences owned by columns of the truncated table.
options.transaction	Transaction	optional	Transaction to run query under
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).

Return:

Promise<Integer> The number of destroyed rows

public static drop(options: Object): Promise

Drop the table represented by this Model

Params:

Name	Type	Attribute	Description
options	Object	optional	
options.cascade	Boolean	optional default: false	Also drop all objects depending on this table, such as views. Only works in postgres
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).

Return:

Promise

public static findAll(options: Object): Promise<Array<Model>>

Search for multiple instances.

Simple search using AND and =

```
Model.findAll({
  where: {
    attr1: 42,
    attr2: 'cake'
  }
})
```

Using greater than, less than etc.

```
const {gt, lte, ne, in: opIn} = Sequelize.Op;
Model.findAll({
  where: {
    attr1: {
      [gt]: 50
    },
    attr2: {
      [lte]: 45
    },
    attr3: {
      [opIn]: [1,2,3]
    },
    attr4: {
      [ne]: 5
    }
  }
})
```

```
WHERE attr1 > 50 AND attr2 <= 45 AND attr3 IN (1,2,3) AND attr4 != 5
```

See Operators for possible operators

Queries using OR

```
const {or, and, gt, lt} = Sequelize.Op;
Model.findAll({
  where: {
    name: 'a project',
    [or]: [
      {id: [1, 2, 3]},
      {
        [and]: [
          {id: {[gt]: 10}},
          {id: {[lt]: 100}}
        ]
      }
    ]
  }
});
```

```
WHERE `Model`.`name` = 'a project' AND (`Model`.`id` IN (1, 2, 3) OR (`Model`.`id` > 10 AND `Model`.`id` < 100));
```

The promise is resolved with an array of Model instances if the query succeeds.

Alias: *all*

Params:

Name	Type	Attribute	Description
options	Object	optional	A hash of options to describe the scope of the search
options.where	Object	optional	A hash of attributes to describe your search. See above for examples.
options.attributes	Array<String> | Object	optional	A list of the attributes that you want to select, or an object with `include` and `exclude` keys. To rename an attribute, you can pass an array, with two elements - the first is the name of the attribute in the DB (or some kind of expression such as `Sequelize.literal`, `Sequelize.fn` and so on), and the second is the name you want the attribute to have in the returned instance

options.attributes.include	Array<String>	optional	Select all the attributes of the model, plus some additional ones. Useful for aggregations, e.g. `{ attributes: { include: [[sequelize.fn('COUNT', sequelize.col('id')), 'total']] }`
options.attributes.exclude	Array<String>	optional	Select all the attributes of the model, except some few. Useful for security purposes e.g. `{ attributes: { exclude: ['password'] } }`
options.paranoid	Boolean	optional default: true	If true, only non-deleted records will be returned. If false, both deleted and non-deleted records will be returned. Only applies if `options.paranoid` is true for the model.
options.include	Array<Object|Model|String>	optional	A list of associations to eagerly load using a left join. Supported is either `{ include: [ Model1, Model2, ...]}` or `{ include: [{ model: Model1, as: 'Alias' }]}` or `{ include: ['Alias']}`. If your association are set up with an `as` (eg. `X.hasMany(Y, { as: 'Z }`, you need to specify Z in the as attribute when eager loading Y).
options.include[].model	Model	optional	The model you want to eagerly load
options.include[].as	String	optional	The alias of the relation, in case the model you want to eagerly load is aliased. For `hasOne` / `belongsTo`, this should be the singular name, and for `hasMany`, it should be the plural
options.include[].association	Association	optional	The association you want to eagerly load. (This can be used instead of providing a model/as pair)
options.include[].where	Object	optional	Where clauses to apply to the child models. Note that this converts the eager load to an inner join, unless you explicitly set `required: false`
options.include[].or	Boolean	optional default: false	Whether to bind the ON and WHERE clause together by OR instead of AND.
options.include[].on	Object	optional	Supply your own ON condition for the join.
options.include[].attributes	Array<String>	optional	A list of attributes to select from the child model
options.include[].required	Boolean	optional	If true, converts to an inner join, which means that the parent model will only be loaded if it has any matching children. True if `include.where` is set, false otherwise.

options.include[].separate	Boolean	optional	If true, runs a separate query to fetch the associated instances, only supported for hasMany associations
options.include[].limit	Number	optional	Limit the joined rows, only supported with include.separate=true
options.include[].through.where	Object	optional	Filter on the join model for belongsToMany relations
options.include[].through.attributes	Array	optional	A list of attributes to select from the join model for belongsToMany relations
options.include[].include	Array<Object|Model|String>	optional	Load further nested related models
options.order	Array | fn | col | literal	optional	Specifies an ordering. Using an array, you can provide several columns / functions to order by. Each element can be further wrapped in a two-element array. The first element is the column / function to order by, the second is the direction. For example: `order: [['name', 'DESC']]`. In this way the column will be escaped, but the direction will not.
options.limit	Number	optional	
options.offset	Number	optional	
options.transaction	Transaction	optional	Transaction to run query under
options.lock	String | Object	optional	Lock the selected rows. Possible options are transaction.LOCK.UPDATE and transaction.LOCK.SHARE. Postgres also supports transaction.LOCK.KEY_SHARE, transaction.LOCK.NO_KEY_UPDATE and specific model locks with joins. See transaction.LOCK for an example
options.raw	Boolean	optional	Return raw result. See sequelize.query for more information.
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).
options.having	Object	optional	
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)

options.rejectOnEmpty	Boolean | Error	optional default: false	Throws an error when no records found

Return:
Promise<Array<Model>>

public static findAndCount(findOptions: Object): Promise<{count: Integer, rows: Model[]}>

Find all the rows matching your query, within a specified offset / limit, and get the total number of rows matching your query. This is very useful for paging

```
Model.findAndCountAll({
  where: ...,
  limit: 12,
  offset: 12
}).then(result => {
  ...
})
```

In the above example, `result.rows` will contain rows 13 through 24, while `result.count` will return the total number of rows that matched your query.

When you add includes, only those which are required (either because they have a where clause, or because `required` is explicitly set to true on the include) will be added to the count part.

Suppose you want to find all users who have a profile attached:

```
User.findAndCountAll({
  include: [
     { model: Profile, required: true}
  ],
  limit 3
});
```

Because the include for `Profile` has `required` set it will result in an inner join, and only the users who have a profile will be counted. If we remove `required` from the include, both users with and without profiles will be counted

Alias: *findAndCountAll*

Params:

Name	Type	Attribute	Description
findOptions	Object	optional	See findAll

Return:
Promise<{count: Integer, rows: Model[]}>

See:
Model.findAll for a specification of find and query options

public static findById(id: Number | String | Buffer, options: Object): Promise<Model>

Search for a single instance by its primary key.

Alias: *findByPrimary*

Params:

Name	Type	Attribute	Description
id	Number | String | Buffer		The value of the desired instance's primary key.
options	Object	optional	
options.transaction	Transaction	optional	Transaction to run query under

options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)

Return:
Promise<Model>

See:
Model.findAll for a full explanation of options

public static findCreateFind(options: Object): Promise<Model, created>

A more performant findOrCreate that will not work under a transaction (at least not in postgres) Will execute a find call, if empty then attempt to create, if unique constraint then attempt to find again

Params:

Name	Type	Attribute	Description
options	Object		
options.where	Object		where A hash of search attributes.
options.defaults	Object	optional	Default values to use if creating a new instance

Return:
Promise<Model, created>

See:
Model.findAll for a full specification of find and options

public static findOne(options: Object): Promise<Model>

Search for a single instance. This applies LIMIT 1, so the listener will always be called with a single instance.

Alias: *find*

Params:

Name	Type	Attribute	Description
options	Object	optional	A hash of options to describe the scope of the search
options.transaction	Transaction	optional	Transaction to run query under
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)

Return:
Promise<Model>

See:
Model.findAll for an explanation of options

public static findOrBuild(options: Object): Promise<Model, initialized>

Find a row that matches the query, or build (but don't save) the row if none is found. The successful result of the promise will be (instance, initialized) - Make sure to use .spread()

Alias: *findOrInitialize*

Params:

Name	Type	Attribute	Description
options	Object		
options.where	Object		A hash of search attributes.

options.defaults	Object	optional	Default values to use if building a new instance
options.transaction	Object	optional	Transaction to run query under
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).

Return:
Promise<Model, initialized>

public static findOrCreate(options: Object): Promise<Model, created>

Find a row that matches the query, or build and save the row if none is found The successful result of the promise will be (instance, created) - Make sure to use .spread()

If no transaction is passed in the `options` object, a new transaction will be created internally, to prevent the race condition where a matching row is created by another connection after the find but before the insert call. However, it is not always possible to handle this case in SQLite, specifically if one transaction inserts and another tries to select before the first one has committed. In this case, an instance of sequelize. TimeoutError will be thrown instead. If a transaction is created, a savepoint will be created instead, and any unique constraint violation will be handled internally.

Params:

Name	Type	Attribute	Description
options	Object		
options.where	Object		where A hash of search attributes.
options.defaults	Object	optional	Default values to use if creating a new instance
options.transaction	Transaction	optional	Transaction to run query under

Return:
Promise<Model, created>

See:
Model.findAll for a full specification of find and options

public static getTableName(): String | Object

Get the tablename of the model, taking schema into account. The method will return The name as a string if the model has no schema, or an object with `tableName` , `schema` and `delimiter` properties.

Return:
String | Object

public static hasMany(target: Model, options: object): HasMany

Creates a 1:m association between this (the source) and the provided target. The foreign key is added on the target.

Params:

Name	Type	Attribute	Description
target	Model		
options	object	optional	

options.hooks	boolean	optional default: false	Set to true to run before-/afterDestroy hooks when an associated model is deleted because of a cascade. For example if `User.hasOne(Profile, {onDelete: 'cascade', hooks:true})` , the before-/afterDestroy hooks for profile will be called when a user is deleted. Otherwise the profile will be deleted without invoking any hooks
options.as	string | object	optional	The alias of this model. If you provide a string, it should be plural, and will be singularized using node.inflection. If you want to control the singular version yourself, provide an object with `plural` and `singular` keys. See also the `name` option passed to `sequelize.define` . If you create multiple associations between the same tables, you should provide an alias to be able to distinguish between them. If you provide an alias when creating the association, you should provide the same alias when eager loading and when getting associated models. Defaults to the pluralized name of target
options.foreignKey	string | object	optional	The name of the foreign key in the target table or an object representing the type definition for the foreign column (see `Sequelize.define` for syntax). When using an object, you can add a `name` property to set the name of the column. Defaults to the name of source + primary key of source
options.sourceKey	string	optional	The name of the field to use as the key for the association in the source table. Defaults to the primary key of the source table
options.scope	object	optional	A key/value set that will be used for association create and find defaults on the target. (sqlite not supported for N:M)
options.onDelete	string	optional default: 'SET NULL|CASCADE'	SET NULL if foreignKey allows nulls, CASCADE if otherwise
options.onUpdate	string	optional default: 'CASCADE'	
options.constraints	boolean	optional default: true	Should on update and on delete constraints be enabled on the foreign key.

Return:

HasMany

Example:

```
User.hasMany(Profile) // This will add userId to the profile table
```

public static hasOne(target: Model, options: object): HasOne

Creates an association between this (the source) and the provided target. The foreign key is added on the target.

Params:

Name	Type	Attribute	Description
target	Model		
options	object	optional	

options.hooks	boolean	optional default: false	Set to true to run before-/afterDestroy hooks when an associated model is deleted because of a cascade. For example if `User.hasOne(Profile, {onDelete: 'cascade', hooks:true})` , the before-/afterDestroy hooks for profile will be called when a user is deleted. Otherwise the profile will be deleted without invoking any hooks
options.as	string	optional	The alias of this model, in singular form. See also the `name` option passed to `sequelize.define` . If you create multiple associations between the same tables, you should provide an alias to be able to distinguish between them. If you provide an alias when creating the association, you should provide the same alias when eager loading and when getting associated models. Defaults to the singularized name of target
options.foreignKey	string | object	optional	The name of the foreign key in the target table or an object representing the type definition for the foreign column (see `Sequelize.define` for syntax). When using an object, you can add a `name` property to set the name of the column. Defaults to the name of source + primary key of source
options.onDelete	string	optional default: 'SET NULL|CASCADE'	SET NULL if foreignKey allows nulls, CASCADE if otherwise
options.onUpdate	string	optional default: 'CASCADE'	
options.constraints	boolean	optional default: true	Should on update and on delete constraints be enabled on the foreign key.

Return:

HasOne

Example:

```
User.hasOne(Profile) // This will add userId to the profile table
```

public static increment(fields: String | Array | Object, options: Object): Promise<this>

Increment the value of one or more columns. This is done in the database, which means it does not use the values currently stored on the Instance. The increment is done using a `SET column = column + X WHERE foo = 'bar'` query. To get the correct value after an increment into the Instance you should do a reload.

```
// increment number by 1
Model.increment('number', { where: { foo: 'bar' });

// increment number and count by 2
Model.increment(['number', 'count'], { by: 2, where: { foo: 'bar' } });

// increment answer by 42, and decrement tries by 1.
// `by` is ignored, since each column has its own value
Model.increment({ answer: 42, tries: -1}, { by: 2, where: { foo: 'bar' } });
```

Params:

Name	Type	Attribute	Description
fields	String | Array | Object		If a string is provided, that column is incremented by the value of `by` given in options. If an array is provided, the same is true for each column. If and object is provided, each column is incremented by the value given.
options	Object		
options.where	Object		

options.by	Integer	optional default: 1	The number to increment by
options.silent	Boolean	optional default: false	If true, the updatedAt timestamp will not be updated.
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.transaction	Transaction	optional	
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)

Return:

Promise<this>

See:

Model#reload

public static init(attributes: Object, options: Object): Model

Initialize a model, representing a table in the DB, with attributes and options.

The table columns are define by the hash that is given as the second argument. Each attribute of the hash represents a column. A short table definition might look like this:

```
Project.init({
  columnA: {
    type: Sequelize.BOOLEAN,
    validate: {
      is: ['[a-z]','i'],        // will only allow letters
      max: 23,                  // only allow values <= 23
      isIn: {
        args: [['en', 'zh']],
        msg: "Must be English or Chinese"
      }
    },
    field: 'column_a'
    // Other attributes here
  },
  columnB: Sequelize.STRING,
  columnC: 'MY VERY OWN COLUMN TYPE'
}, {sequelize})

sequelize.models.modelName // The model will now be available in models under the class name
```

As shown above, column definitions can be either strings, a reference to one of the datatypes that are predefined on the Sequelize constructor, or an object that allows you to specify both the type of the column, and other attributes such as default values, foreign key constraints and custom setters and getters.

For a list of possible data types, see DataTypes

For more about validation, see http://docs.sequelizejs.com/manual/tutorial/models-definition.html#validations

Params:

Name	Type	Attribute	Description

attributes	Object		An object, where each attribute is a column of the table. Each column can be either a DataType, a string or a type-description object, with the properties described below:
attributes.column	String | DataTypes | Object		The description of a database column
attributes.column.type	String | DataTypes		A string or a data type
attributes.column.allowNull	Boolean	optional default: true	If false, the column will have a NOT NULL constraint, and a not null validation will be run before an instance is saved.
attributes.column.defaultValue	any	optional default: null	A literal default value, a JavaScript function, or an SQL function (see `sequelize.fn`)
attributes.column.unique	String | Boolean	optional default: false	If true, the column will get a unique constraint. If a string is provided, the column will be part of a composite unique index. If multiple columns have the same string, they will be part of the same unique index
attributes.column.primaryKey	Boolean	optional default: false	
attributes.column.field	String	optional default: null	If set, sequelize will map the attribute name to a different name in the database
attributes.column.autoIncrement	Boolean	optional default: false	
attributes.column.comment	String	optional default: null	
attributes.column.references	String | Model	optional default: null	An object with reference configurations

attributes.column.references.model	String | Model	optional	If this column references another table, provide it here as a Model, or a string
attributes.column.references.key	String	optional default: 'id'	The column of the foreign table that this column references
attributes.column.onUpdate	String	optional	What should happen when the referenced key is updated. One of CASCADE, RESTRICT, SET DEFAULT, SET NULL or NO ACTION
attributes.column.onDelete	String	optional	What should happen when the referenced key is deleted. One of CASCADE, RESTRICT, SET DEFAULT, SET NULL or NO ACTION
attributes.column.get	Function	optional	Provide a custom getter for this column. Use `this.getDataValue(String)` to manipulate the underlying values.
attributes.column.set	Function	optional	Provide a custom setter for this column. Use `this.setDataValue(String, Value)` to manipulate the underlying values.

attributes.validate	Object	optional	An object of validations to execute for this column every time the model is saved. Can be either the name of a validation provided by validator.js, a validation function provided by extending validator.js (see the `DAOValidator` property for more details), or a custom validation function. Custom validation functions are called with the value of the field, and can possibly take a second callback argument, to signal that they are asynchronous. If the validator is sync, it should throw in the case of a failed validation, it it is async, the callback should be called with the error text.
options	Object		These options are merged with the default define options provided to the Sequelize constructor
options.sequelize	Object		Define the sequelize instance to attach to the new Model. Throw error if none is provided.
options.modelName	String	optional	Set name of the model. By default its same as Class name.
options.defaultScope	Object	optional default: {}	Define the default search scope to use for this model. Scopes have the same form as the options passed to find / findAll

options.scopes	Object	optional	More scopes, defined in the same way as defaultScope above. See `Model.scope` for more information about how scopes are defined, and what you can do with them
options.omitNull	Boolean	optional	Don't persist null values. This means that all columns with null values will not be saved
options.timestamps	Boolean	optional default: true	Adds createdAt and updatedAt timestamps to the model.
options.paranoid	Boolean	optional default: false	Calling `destroy` will not delete the model, but instead set a `deletedAt` timestamp if this is true. Needs `timestamps=true` to work
options.underscored	Boolean	optional default: false	Converts all camelCased columns to underscored if true. Will not affect timestamp fields named explicitly by model options and will not affect fields with explicitly set `field` option
options.underscoredAll	Boolean	optional default: false	Converts camelCased model names to underscored table names if true. Will not change model name if freezeTableName is set to true

options.freezeTableName	Boolean	optional default: false	If freezeTableName is true, sequelize will not try to alter the model name to get the table name. Otherwise, the model name will be pluralized
options.name	Object	optional	An object with two attributes, `singular` and `plural`, which are used when this model is associated to others.
options.name.singular	String	optional default: Utils.singularize(modelName)	
options.name.plural	String	optional default: Utils.pluralize(modelName)	
options.indexes	Array<Object>	optional	
options.indexes[].name	String	optional	The name of the index. Defaults to model name + _ + fields concatenated
options.indexes[].type	String	optional	Index type. Only used by mysql. One of `UNIQUE`, `FULLTEXT` and `SPATIAL`
options.indexes[].method	String	optional	The method to create the index by (`USING` statement in SQL). BTREE and HASH are supported by mysql and postgres, and postgres additionally supports GIST and GIN.
options.indexes[].unique	Boolean	optional default: false	Should the index by unique? Can also be triggered by setting type to `UNIQUE`
options.indexes[].concurrently	Boolean	optional default: false	PostgreSQL will build the index without taking any write locks. Postgres only

options.indexes[].fields	Array<String|Object>	optional	An array of the fields to index. Each field can either be a string containing the name of the field, a sequelize object (e.g `sequelize.fn`), or an object with the following attributes: `attribute` (field name), `length` (create a prefix index of length chars), `order` (the direction the column should be sorted in), `collate` (the collation (sort order) for the column)
options.createdAt	String | Boolean	optional	Override the name of the createdAt column if a string is provided, or disable it if false. Timestamps must be true. Not affected by underscored setting.
options.updatedAt	String | Boolean	optional	Override the name of the updatedAt column if a string is provided, or disable it if false. Timestamps must be true. Not affected by underscored setting.
options.deletedAt	String | Boolean	optional	Override the name of the deletedAt column if a string is provided, or disable it if false. Timestamps must be true. Not affected by underscored setting.

options.tableName	String	optional	Defaults to pluralized model name, unless freezeTableName is true, in which case it uses model name verbatim
options.schema	String	optional default: 'public'	
options.engine	String	optional	
options.charset	String	optional	
options.comment	String	optional	
options.collate	String	optional	
options.initialAutoIncrement	String	optional	Set the initial AUTO_INCREMENT value for the table in MySQL.
options.hooks	Object	optional	An object of hook function that are called before and after certain lifecycle events. The possible hooks are: beforeValidate, afterValidate, validationFailed, beforeBulkCreate, beforeBulkDestroy, beforeBulkUpdate, beforeCreate, beforeDestroy, beforeUpdate, afterCreate, afterDestroy, afterUpdate, afterBulkCreate, afterBulkDestory and afterBulkUpdate. See Hooks for more information about hook functions and their signatures. Each property can either be a function, or an array of functions.

options.validate	Object	optional	An object of model wide validations. Validations have access to all model values via `this`. If the validator function takes an argument, it is assumed to be async, and is called with a callback that accepts an optional error.

Return:
Model

See:
DataTypes
Hooks

public static max(field: String, options: Object): Promise<Any>

Find the maximum value of field

Params:

Name	Type	Attribute	Description
field	String		
options	Object	optional	See aggregate

Return:
Promise<Any>

See:
Model#aggregate for options

public static min(field: String, options: Object): Promise<Any>

Find the minimum value of field

Params:

Name	Type	Attribute	Description
field	String		
options	Object	optional	See aggregate

Return:
Promise<Any>

See:
Model#aggregate for options

public static removeAttribute(attribute: String)

Remove attribute from model definition

Params:

Name	Type	Attribute	Description
attribute	String	optional	

Restore multiple instances if `paranoid` is enabled.

Params:

Name	Type	Attribute	Description
options	Object		
options.where	Object	optional	Filter the restore
options.hooks	Boolean	optional default: true	Run before / after bulk restore hooks?
options.individualHooks	Boolean	optional default: false	If set to true, restore will find all records within the where parameter and will execute before / after bulkRestore hooks on each row
options.limit	Number	optional	How many rows to undelete (only for mysql)
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).
options.transaction	Transaction	optional	Transaction to run query under

Return:
Promise<undefined>

public static schema(schema: String, options: Object): this

Apply a schema to this model. For postgres, this will actually place the schema in front of the table name - `"schema"."tableName"` , while the schema will be prepended to the table name for mysql and sqlite - `'schema.tablename'` .

This method is intended for use cases where the same model is needed in multiple schemas. In such a use case it is important to call `model.schema(schema, [options]).sync()` for each model to ensure the models are created in the correct schema.

If a single default schema per model is needed, set the `options.schema='schema'` parameter during the `define()` call for the model.

Params:

Name	Type	Attribute	Description
schema	String		The name of the schema
options	Object	optional	
options.schemaDelimiter	String	optional default: '.'	The character(s) that separates the schema name from the table name
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).

Return:
this

See:

public static scope(options: Array | Object | String | null): Model

Apply a scope created in `define` to the model. First let's look at how to create scopes:

```
const Model = sequelize.define('model', attributes, {
  defaultScope: {
    where: {
      username: 'dan'
    },
    limit: 12
  },
  scopes: {
    isALie: {
      where: {
        stuff: 'cake'
      }
    },
    complexFunction: function(email, accessLevel) {
      return {
        where: {
          email: {
            [Op.like]: email
          },
          accesss_level {
            [Op.gte]: accessLevel
          }
        }
      }
    }
  }
})
```

Now, since you defined a default scope, every time you do Model.find, the default scope is appended to your query. Here's a couple of examples:

```
Model.findAll() // WHERE username = 'dan'
Model.findAll({ where: { age: { [Op.gt]: 12 } } }) // WHERE age > 12 AND username = 'dan'
```

To invoke scope functions you can do:

```
Model.scope({ method: ['complexFunction', 'dan@sequelize.com', 42]}).findAll()
// WHERE email like 'dan@sequelize.com%' AND access_level >= 42
```

Params:

Name	Type	Attribute	Description
options	Array | Object | String | null		The scope(s) to apply. Scopes can either be passed as consecutive arguments, or as an array of arguments. To apply simple scopes and scope functions with no arguments, pass them as strings. For scope function, pass an object, with a `method` property. The value can either be a string, if the method does not take any arguments, or an array, where the first element is the name of the method, and consecutive elements are arguments to that method. Pass null to remove all scopes, including the default.

Return:

Model	A reference to the model, with the scope(s) applied. Calling scope again on the returned model will clear the previous scope.

public static sum(field: String, options: Object): Promise<Number>

Find the sum of field

Params:

Name	Type	Attribute	Description
field	String		
options	Object	optional	See aggregate

Return:

Promise<Number>

public static sync(options: *): Promise<this>

Sync this Model to the DB, that is create the table. Upon success, the callback will be called with the model instance (this)

Params:

Name	Type	Attribute	Description
options	*		

Return:
Promise<this>

See:
Sequelize#sync for options

public static truncate(options: object): Promise

Truncate all instances of the model. This is a convenient method for Model.destroy({ truncate: true }).

Params:

Name	Type	Attribute	Description
options	object	optional	The options passed to Model.destroy in addition to truncate
options.cascade	Boolean | function	optional default: false	Only used in conjunction with TRUNCATE. Truncates all tables that have foreign-key references to the named table, or to any tables added to the group due to CASCADE.
options.transaction	Transaction	optional	Transaction to run query under
options.logging	Boolean | function	optional	A function that logs sql queries, or false for no logging
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)

Return:
Promise

See:
Model#destroy for more information

public static unscoped(): Model

Return:
Model

public static update(values: Object, options: Object): Promise<Array<affectedCount, affectedRows>>

Update multiple instances that match the where options. The promise returns an array with one or two elements. The first element is always the number of affected rows, while the second element is the actual affected rows (only supported in postgres with `options.returning` true.)

Params:

Name	Type	Attribute	Description
values	Object		

options	Object		
options.where	Object		Options to describe the scope of the search.
options.paranoid	Boolean	optional default: true	If true, only non-deleted records will be updated. If false, both deleted and non-deleted records will be updated. Only applies if `options.paranoid` is true for the model.
options.fields	Array	optional	Fields to update (defaults to all fields)
options.validate	Boolean	optional default: true	Should each row be subject to validation before it is inserted. The whole insert will fail if one row fails validation
options.hooks	Boolean	optional default: true	Run before / after bulk update hooks?
options.sideEffects	Boolean	optional default: true	Whether or not to update the side effects of any virtual setters.
options.individualHooks	Boolean	optional default: false	Run before / after update hooks?. If true, this will execute a SELECT followed by individual UPDATEs. A select is needed, because the row data needs to be passed to the hooks
options.returning	Boolean	optional default: false	Return the affected rows (only for postgres)
options.limit	Number	optional	How many rows to update (only for mysql and mariadb, implemented as TOP(n) for MSSQL; for sqlite it is supported only when rowid is present)
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).
options.transaction	Transaction	optional	Transaction to run query under
options.silent	Boolean	optional default: false	If true, the updatedAt timestamp will not be updated.

Return:

Promise<Array<affectedCount, affectedRows>>

public static upsert(values: Object, options: Object): Promise<created>

Insert or update a single row. An update will be executed if a row which matches the supplied values on either the primary key or a unique key is found. Note that the unique index must be defined in your sequelize model and not just in the table. Otherwise you may experience a unique constraint violation, because sequelize fails to identify the row that should be updated.

Implementation details:

- MySQL - Implemented as a single query `INSERT values ON DUPLICATE KEY UPDATE values`
- PostgreSQL - Implemented as a temporary function with exception handling: INSERT EXCEPTION WHEN unique_constraint UPDATE
- SQLite - Implemented as two queries `INSERT; UPDATE`. This means that the update is executed regardless of whether the row already existed or not

OR IGNORE + UPDATE, in a single query, so there is no way to know whether the row was inserted or not.

Alias: *insertOrUpdate*

Params:

Name	Type	Attribute	Description
values	Object		
options	Object	optional	
options.validate	Boolean	optional default: true	Run validations before the row is inserted
options.fields	Array	optional default: Object.keys(this.attributes)	The fields to insert / update. Defaults to all changed fields
options.hooks	Boolean	optional default: true	Run before / after upsert hooks?
options.returning	Boolean	optional default: false	Append RETURNING * to get back auto generated values (Postgres only)
options.transaction	Transaction	optional	Transaction to run query under
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.benchmark	Boolean	optional default: false	Pass query execution time in milliseconds as second argument to logging function (options.logging).
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)

Return:

Promise<created>	Returns a boolean indicating whether the row was created or updated. For Postgres/MSSQL with (options.returning=true), it returns record and created boolean with signature `<Model, created>` .

Public Constructors

public constructor(values: Object, options: Object)

Builds a new model instance.

Params:

Name	Type	Attribute	Description
values	Object	optional default: {}	an object of key value pairs
options	Object	optional	
options.raw	Boolean	optional default: false	If set to true, values will ignore field and virtual setters.
options.isNewRecord	Boolean	optional default: true	
options.include	Array	optional	an array of include options - Used to build prefetched/included model instances. See `set`

public isNewRecord: Boolean: *

Returns true if this instance has not yet been persisted to the database

Properties:

Name	Type	Attribute	Description
isNewRecord	*		

Return:
Boolean

Return Properties:

Name	Type	Attribute	Description
isNewRecord	*		

public get sequelize: Sequelize: *

A reference to the sequelize instance

Properties:

Name	Type	Attribute	Description
sequelize	*		

Return:
Sequelize

Return Properties:

Name	Type	Attribute	Description
sequelize	*		

See:
Sequelize

Public Methods

public changed(key: String): Boolean | Array

If changed is called with a string it will return a boolean indicating whether the value of that key in `dataValues` is different from the value in `_previousDataValues` .

If changed is called without an argument, it will return an array of keys that have changed.

If changed is called without an argument and no keys have changed, it will return `false` .

Params:

Name	Type	Attribute	Description
key	String	optional	

Return:
Boolean | Array

public decrement(fields: String | Array | Object, options: Object): Promise

Decrement the value of one or more columns. This is done in the database, which means it does not use the values currently stored on the Instance. The decrement is done using a

```
SET column = column - X
```

query. The updated instance will be returned by default in Postgres. However, in other dialects, you will need to do a reload to get the new values.

```
instance.decrement({ answer: 42, tries: 1}, { by: 2 }) // decrement answer by 42, and tries by 1.
                                                       // `by` is ignored, since each column has its own value
```

Params:

Name	Type	Attribute	Description
fields	String | Array | Object		If a string is provided, that column is decremented by the value of `by` given in options. If an array is provided, the same is true for each column. If and object is provided, each column is decremented by the value given
options	Object	optional	
options.by	Integer	optional default: 1	The number to decrement by
options.silent	Boolean	optional default: false	If true, the updatedAt timestamp will not be updated.
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.transaction	Transaction	optional	
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)
options.returning	Boolean	optional default: true	Append RETURNING * to get back auto generated values (Postgres only)

Return:
Promise

See:
Model#reload

public destroy(options: Object): Promise<undefined>

Destroy the row corresponding to this instance. Depending on your setting for paranoid, the row will either be completely deleted, or have its deletedAt timestamp set to the current time.

Params:

Name	Type	Attribute	Description
options	Object	optional default: {}	
options.force	Boolean	optional default: false	If set to true, paranoid models will actually be deleted
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.transaction	Transaction	optional	
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)

Return:
Promise<undefined>

Check whether this and `other` Instance refer to the same row

Params:

Name	Type	Attribute	Description
other	Model		

Return:
Boolean

public equalsOneOf(others: Array): Boolean

Check if this is equal to one of `others` by calling equals

Params:

Name	Type	Attribute	Description
others	Array		

Return:
Boolean

public get(key: String, options: Object): Object | any

If no key is given, returns all values of the instance, also invoking virtual getters.

If key is given and a field or virtual getter is present for the key it will call that getter - else it will return the value for key.

Params:

Name	Type	Attribute	Description
key	String	optional	
options	Object	optional	
options.plain	Boolean	optional default: false	If set to true, included instances will be returned as plain objects
options.raw	Boolean	optional default: false	If set to true, field and virtual setters will be ignored

Return:
Object | any

public getDataValue(key: String): any

Get the value of the underlying data value

Params:

Name	Type	Attribute	Description
key	String		

Return:
any

public increment(fields: String | Array | Object, options: Object): Promise<this>

since 4.0.0

Increment the value of one or more columns. This is done in the database, which means it does not use the values currently stored on the Instance. The increment is done using a

```
SET column = column + X
```

```
instance.increment('number') // increment number by 1
instance.increment(['number', 'count'], { by: 2 }) // increment number and count by 2
instance.increment({ answer: 42, tries: 1}, { by: 2 }) // increment answer by 42, and tries by 1.
                                                     // `by` is ignored, since each column has its own value
```

Params:

Name	Type	Attribute	Description
fields	String | Array | Object		If a string is provided, that column is incremented by the value of `by` given in options. If an array is provided, the same is true for each column. If and object is provided, each column is incremented by the value given.
options	Object	optional	
options.by	Integer	optional default: 1	The number to increment by
options.silent	Boolean	optional default: false	If true, the updatedAt timestamp will not be updated.
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.transaction	Transaction	optional	
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)
options.returning	Boolean	optional default: true	Append RETURNING * to get back auto generated values (Postgres only)

Return:
Promise<this>

See:
Model#reload

public isSoftDeleted(): Boolean

Helper method to determine if a instance is "soft deleted". This is particularly useful if the implementer renamed the `deletedAt` attribute to something different. This method requires `paranoid` to be enabled.

Return:
Boolean

public previous(key: String): any | Array<any>

Returns the previous value for key from `_previousDataValues`.

If called without a key, returns the previous values for all values which have changed

Params:

Name	Type	Attribute	Description
key	String	optional	

Return:
any | Array<any>

Refresh the current instance in-place, i.e. update the object with current data from the DB and return the same object. This is different from doing a `find(Instance.id)` , because that would create and return a new instance. With this method, all references to the Instance are updated with the new data and no new objects are created.

Params:

Name	Type	Attribute	Description
options	Object	optional	Options that are passed on to `Model.find`
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.

Return:
Promise<this>

See:
Model.findAll

public restore(options: Object): Promise<undefined>

Restore the row corresponding to this instance. Only available for paranoid models.

Params:

Name	Type	Attribute	Description
options	Object	optional default: {}	
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.transaction	Transaction	optional	

Return:
Promise<undefined>

public save(options: Object): Promise<this|Errors.ValidationError>

Validate this instance, and if the validation passes, persist it to the database. It will only save changed fields, and do nothing if no fields have changed.

On success, the callback will be called with this instance. On validation error, the callback will be called with an instance of `Sequelize.ValidationError` . This error will have a property for each of the fields for which validation failed, with the error message for that field.

Params:

Name	Type	Attribute	Description
options	Object	optional	
options.fields	string[]	optional	An optional array of strings, representing database columns. If fields is provided, only those columns will be validated and saved.
options.silent	Boolean	optional default: false	If true, the updatedAt timestamp will not be updated.
options.validate	Boolean	optional default: true	If false, validations won't be run.

options.hooks	Boolean	optional default: true	Run before and after create / update + validate hooks
options.logging	Function	optional default: false	A function that gets executed while running the query to log the sql.
options.transaction	Transaction	optional	
options.searchPath	String	optional default: DEFAULT	An optional parameter to specify the schema search_path (Postgres only)
options.returning	Boolean	optional	Append RETURNING * to get back auto generated values (Postgres only)

Return:

Promise<this|Errors.ValidationError>

public set(key: String | Object, value: any, options: Object): *

Set is used to update values on the instance (the sequelize representation of the instance that is, remember that nothing will be persisted before you actually call `save`). In its most basic form `set` will update a value stored in the underlying `dataValues` object. However, if a custom setter function is defined for the key, that function will be called instead. To bypass the setter, you can pass `raw: true` in the options object.

If set is called with an object, it will loop over the object, and call set recursively for each key, value pair. If you set raw to true, the underlying dataValues will either be set directly to the object passed, or used to extend dataValues, if dataValues already contain values.

When set is called, the previous value of the field is stored and sets a changed flag(see `changed`).

Set can also be used to build instances for associations, if you have values for those. When using set with associations you need to make sure the property key matches the alias of the association while also making sure that the proper include options have been set (from .build() or .find())

If called with a dot.separated key on a JSON/JSONB attribute it will set the value nested and flag the entire object as changed.

Params:

Name	Type	Attribute	Description
key	String | Object		
value	any		
options	Object	optional	
options.raw	Boolean	optional default: false	If set to true, field and virtual setters will be ignored
options.reset	Boolean	optional default: false	Clear all previously set data values

Return:

*

See:

Model.findAll for more information about includes

public setDataValue(key: String, value: any)

Update the underlying data value

Params:

key	String		
value	any		

public toJSON(): object

Convert the instance to a JSON representation. Proxies to calling `get` with no keys. This means get all values gotten from the DB, and apply all custom getters.

Return:
object

See:
Model#get

public update(updates: Object, options: Object): Promise<this>

This is the same as calling `set` and then calling `save` but it only saves the exact values passed to it, making it more atomic and safer.

Params:

Name	Type	Attribute	Description
updates	Object		See `set`
options	Object		See `save`

Return:
Promise<this>

See:
Model#set
Model#save

public validate(options: Object): Promise<undefined>

Validate the attributes of this instance according to validation rules set in the model definition.

The promise fulfills if and only if validation successful; otherwise it rejects an Error instance containing { field name : [error msgs] } entries.

Params:

Name	Type	Attribute	Description
options	Object	optional	Options that are passed to the validator
options.skip	Array	optional	An array of strings. All properties that are in this array will not be validated
options.fields	Array	optional	An array of strings. Only the properties that are in this array will be validated
options.hooks	Boolean	optional default: true	Run before and after validate hooks

Return:
Promise<undefined>

public where(checkVersion: *): Object

Get an object representing the query for this instance, use with `options.where`

Params:

Name	Type	Attribute	Description

checkVersion	*		

Return:

Object

Return Properties:

Name	Type	Attribute	Description
where	*		

Generated by ESDoc(0.5.2)