

## Table of Contents

<b>POSTGRESQL DATABASE OBJECT MANAGEMENT</b>	<b>4</b>
POSTGRESQL SCHEMAS	5
<i>PostgreSQL Schema Designer</i>	7
Editing PostgreSQL Schema General	8
<i>PostgreSQL Tables</i>	9
PostgreSQL Table Designer	14
PostgreSQL Table Fields	15
Setting PostgreSQL Table Field Properties	17
Setting Other PostgreSQL Table Field Properties	20
PostgreSQL Table Indexes	21
Setting PostgreSQL Table Index Properties	22
PostgreSQL Table Foreign Keys	23
Setting PostgreSQL Table Foreign Key Properties	24
PostgreSQL Table Uniques	25
Setting PostgreSQL Table Unique Properties	26
PostgreSQL Table Checks	27
Setting PostgreSQL Table Check Properties	28
PostgreSQL Table Excludes	29
Setting PostgreSQL Table Exclude Properties	30
PostgreSQL Table Rules	31
Setting PostgreSQL Table Rule Properties	32
PostgreSQL Table Triggers	33
Setting PostgreSQL Table Trigger Properties	34
PostgreSQL Table Options	36
<i>PostgreSQL Views</i>	37
PostgreSQL View Designer	41
Working with PostgreSQL View Builder (Available only in Full Version)	42
Editing PostgreSQL View SQL Definition	43
Setting Advanced PostgreSQL View Properties	44
PostgreSQL View Preview	45
PostgreSQL View Explain	46
PostgreSQL View Viewer	47
<i>PostgreSQL Functions</i>	48
PostgreSQL Function Wizard	52
Setting Parameters for PostgreSQL Function	53
Setting Return Type for PostgreSQL Function	54
PostgreSQL Function Designer	55

Editing PostgreSQL Function Definition	56
Setting Advanced PostgreSQL Function Properties	57
Viewing PostgreSQL Function Result	59
<i>PostgreSQL Aggregates</i>	<i>60</i>
PostgreSQL Aggregate Designer	62
Editing PostgreSQL Aggregate Properties	63
<i>PostgreSQL Conversions</i>	<i>64</i>
PostgreSQL Conversion Designer	66
Editing PostgreSQL Conversion Properties	67
<i>PostgreSQL Domains</i>	<i>68</i>
PostgreSQL Domain Designer	70
Editing PostgreSQL Domain General	71
Editing PostgreSQL Domain Check	72
<i>PostgreSQL Trigger Functions</i>	<i>73</i>
PostgreSQL Trigger Function Designer	75
Editing PostgreSQL Trigger Function Definition	76
Setting Advanced PostgreSQL Trigger Function Properties	77
<i>PostgreSQL Operators</i>	<i>79</i>
PostgreSQL Operator Designer	81
Editing PostgreSQL Operator General	82
Editing Advanced PostgreSQL Operator Properties	83
<i>PostgreSQL Operator Classes</i>	<i>84</i>
PostgreSQL Operator Class Designer	86
Editing PostgreSQL Operator Class General	87
Editing PostgreSQL Operator Class Operators	88
Editing PostgreSQL Operator Class Functions	89
<i>PostgreSQL Sequences</i>	<i>90</i>
PostgreSQL Sequence Designer	92
Editing PostgreSQL Sequence General	93
<i>PostgreSQL Types</i>	<i>95</i>
PostgreSQL Type Designer	97
Editing PostgreSQL Base Type Properties	98
Editing PostgreSQL Base Type General	99
Editing Advanced PostgreSQL Base Type Properties	100
Editing PostgreSQL Composite Type Properties	101
Editing PostgreSQL Composite Type General	102
Editing PostgreSQL Enum Type Properties	103
Editing PostgreSQL Enum Type General	104

POSTGRESQL TABLESPACES	105
<i>PostgreSQL Tablespace Designer</i>	107
Editing PostgreSQL Tablespace General	108
POSTGRESQL CASTS	109
<i>PostgreSQL Cast Designer</i>	111
Editing PostgreSQL Cast General	112
POSTGRESQL LANGUAGES	113
<i>PostgreSQL Language Designer</i>	115
Editing PostgreSQL Language General	116

## PostgreSQL Database Object Management

The following list contains the most common PostgreSQL database objects supported by Navicat.

- [Schemas](#)
- [Tables](#)
- [Views](#)
- [Functions](#)
- [Aggregates](#)
- [Conversions](#)
- [Domains](#)
- [Trigger Functions](#)
- [Operators](#)
- [Operator Class](#)
- [Sequences](#)
- [Types](#)
- [Tablespaces](#)
- [Casts](#)
- [Languages](#)



## PostgreSQL Schemas

A schema is essentially a namespace: it contains named objects (tables, data types, functions, and operators) whose names may duplicate those of other objects existing in other schemas.

The schema name must be distinct from any existing schema name in the current database.


### Create Schema

To create a new schema

- Right-click the database in the navigation pane and choose  **New Schema....**  
or
- Right-click any existing schema and choose  **New Schema....**
- Edit schema properties on the appropriate tabs of the Schema Designer.


### Edit Schema

To edit the existing schema (manage its general etc)

- Right-click the schema in the navigation pane and choose  **Schema Properties....**
- Edit schema properties on the appropriate tabs of the Schema Designer.


### Open Schema

To open a schema which shows in the navigation pane

- Double-click the schema to open in the navigation pane.  
or
- Right-click the schema and choose  **Open Schema.**


### Close Schema

To close a schema

- Right-click the schema in the navigation pane and choose  **Close Schema.**

## Delete Schema

To delete a schema

- Right-click the schema in the navigation pane and choose  **Delete Schema**.
- Confirm deleting in the dialog window.

## PostgreSQL Schema Designer

**Schema Designer** is the basic Navicat tool for working with schema. It allows you to create new schema and edit the existing schema properties.

- [Editing Schema General](#)
- Editing Schema Comment

## Editing PostgreSQL Schema General

### **Schema Name**


The name of a schema to be created. The name cannot begin with `pg_`, as such names are reserved for system schemas.

### **Owner**

The name of the user who will own the schema. If omitted, defaults to the user executing the command.



## PostgreSQL Tables


Relational databases use tables to store data. All operations on data are done on the tables themselves or produce another tables as the result. A table is a set of rows and columns, and their intersections are fields. From a general perspective, columns within a table describe the name and type of data that will be found by row for that column's fields. Rows within a table represent records composed of fields that are described from left to right by their corresponding column's name and type. Each field in a row is implicitly correlated with each other field in that row.

Just simply click  to open an object pane for **Table**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit, open and delete the selected table.

### Create Table

To create a new table

- Select anywhere on the object pane.
- Click the  **New Table** from the object pane toolbar.  
or
- Right-click and select  **New Table** from the popup menu.
- Edit table properties and fields on the appropriate tabs of the Table Designer.

**Hint:** To create new table you can also right-click the Tables node of the navigation pane and select the  **New Table** from the popup menu.

To create a new table with the same properties as one of the existing tables has (using popup menu)

**Apply to:** current database {same connection}

- Select the table(s) for copying in the navigation pane/object pane.
- Right-click and select the **Duplicate Table** from the popup menu.
- The newly created table(s) will be named as "tablename\_**copy**".

To create a new table with the same properties as one of the existing tables has (using drag and drop method)

**Apply to:** current database {same connection}




- Select the table(s) for copying in the navigation pane/object pane.
- Right-click and drag the chosen table(s) to the target location.
- Select one of the following options:
  - Copy here (Structure and Data)
  - Copy here (Structure only)
  - Move here
  - Cancel
- The newly created table(s) will be named as "tablename\_**copy**"

**Apply to:** different database {same connection}

different database {different connection (same or cross server type)} (Data Transfer tool will be activated)

- Select the table(s) for copying in the object pane.
- Drag and drop the chosen table(s) to the target database.
- Select one of the following options:
  - Copy here (Structure and Data)
  - Copy here (Structure only)
  - Cancel

To create a new table with modification as one of the existing tables

- Select the table for modifying in the navigation pane/object pane.
- Right-click and select the  **Design Table** from the popup menu.  
or
- Click the  **Design Table** from the object pane toolbar.
- Modify table properties and fields on the appropriate tabs of the Table Designer.
- Click  **Save As**.

## Create Table Shortcut



To create a table shortcut

- Select the table for editing in the navigation pane/object pane.
- Right-click and select **Create Open Table Shortcut...** from the popup menu.
- Define the location you wish your shortcut to be saved.

**Note:** This option is used to provide a convenient way for you to open your table for entering data directly (Grid View/Form View) without activating the main Navicat.

## Edit Table

To edit the existing table (manage its fields, indexes, foreign keys and triggers etc)



- Select the table for editing in the navigation pane/object pane.
- Right-click and select the  **Design Table** from the popup menu.  
or
- Click the  **Design Table** from the object pane toolbar.
- Edit table properties and fields on the appropriate tabs of the Table Designer.


To change the name of the table

- Select the table for editing in the navigation pane/object pane.
- Right-click and select the **Rename** from the popup menu.


## Open Table (manage table data)

To open a table

- Select the table for opening in the navigation pane/object pane.
- Right-click and select the  **Open Table** from the popup menu or simply double-click the table.  
or
- Click the  **Open Table** from the object pane toolbar.

**Note:** This option is only applied if you do wish Navicat loads all your images while opening the table. To open the graphical table with faster performance, use  **Open Table (Quick)** below.

To open a table with graphical fields

- Select the table for opening in the navigation pane/object pane.
- Right-click and select the  **Open Table (Quick)** from the popup menu.

**Note:** Faster performance for opening the graphical table, as BLOB fields (images) will not be loaded until you click on the cell.

## Empty Table

To empty a table

- Select the table in the navigation pane/object pane.
- Right-click the selected table and choose **Empty Table** from the popup menu.

**Note:** This option is only applied when you wish to clear all the existing records without resetting the auto-increment value. To reset the auto-increment value while emptying your table, use **Truncate Table** below.



## Truncate Table

To truncate a table

- Select the table in the navigation pane/object pane.
- Right-click the selected table and choose **Truncate Table** from the popup menu.

## Delete Table

To delete a table

- Select the table for deleting in the navigation pane/object pane.
- Right-click and select the  **Delete Table** from the popup menu.  
or
- Click the  **Delete Table** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Table Information

To achieve a table information

- Select the table in the object pane.
- Right-click the selected table and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Table Designer

**Table Designer** is the basic Navicat tool for working with tables. It allows you to create, edit and drop table's fields, indexes, foreign keys, and much more.



- [Managing Table Fields](#)
- [Managing Table Indexes](#)
- [Managing Table Foreign Keys](#)
- [Managing Table Uniques](#)
- [Managing Table Checks](#)
- [Managing Table Excludes](#)
- [Managing Table Rules](#)
- [Managing Table Triggers](#)
- [Managing Table Options](#)
- Managing Table Comment
- Table SQL Preview

## PostgreSQL Table Fields

Table fields are managed on the **Fields** tab of the Table Designer. Just simply click a field for editing. A right-click displays the popup menu or using the field toolbar, allowing you to create new and drop the selected field.

### Add Field

To add a field to the table

- Open the table in the Table Designer.
- Open the **Fields** tab.
- Right-click and select the  **Add Field** from the popup menu or click the  **Add Field** from the toolbar.
- Edit field properties.

To add a new field with modification as one of the existing fields

- Open the table in the Table Designer.
- Open the **Fields** tab.
- Select field.
- Right-click and select the **Duplicate Field** from the popup menu.
- Edit field properties.



### Edit Field

To edit the table field


- Open the table in the Table Designer.
- Open the **Fields** tab.
- Simply click on the field to edit.

## Delete Field

To delete the table field

- Open the table in the Table Designer.
- Open the **Fields** tab.
- Right-click and select the  **Delete Field** from the popup menu or click the  **Delete Field** from the toolbar.
- Confirm deleting in the dialog window.

## Setting PostgreSQL Table Field Properties

Name	Type	Length	Decimals	Allow Null	
▶ CustNo	float8	53	0	<input type="checkbox"/>	 1
Company	varchar	30	0	<input checked="" type="checkbox"/>	
Addr1	text	0	0	<input checked="" type="checkbox"/>	
Addr2	text	0	0	<input checked="" type="checkbox"/>	

### Name

The Name is a descriptive identifier for a field that can be up to 63 bytes long. The names should be descriptive enough that anyone can easily identify them when viewing or editing records. For example, LastName, FirstName, StreetAddress, or HomePhone.

Use the **Name** edit box to set the field name. Note that the name of the field must be unique among all the field names in the table.

### Type

After you name a field, you choose a data type for the data to be contained in the field. When you choose a field's data type, you are deciding:

- What kind of values to allow in the field. You cannot store text in field with the **Numeric** data type.
- How much storage space PostgreSQL is to set aside for the data in that field.
- What types of operations can be performed on the values in that field.

The **Type** dropdown list defines the type of the field data.

The following table shows the built-in general-purpose data types for PostgreSQL 8.3. Most of the alternative names listed in the "Aliases" column are the names used internally by PostgreSQL for historical reasons.

**Note:** Some built-in general-purpose data types are not applicable for PostgreSQL 8.2 or earlier versions.

Name	Aliases	Description
bigint	int8	signed eight-byte integer
bigserial	serial8	autoincrementing eight-byte integer
bit [ (n) ]		fixed-length bit string

bit varying [ (n) ]	varbit	variable-length bit string
boolean	bool	logical Boolean (true/false)
box		rectangular box in the plane
bytea		binary data ("byte array")
character varying [ (n) ]	varchar [ (n) ]	variable-length character string
character [ (n) ]	char [ (n) ]	fixed-length character string
cidr		IPv4 or IPv6 network address
circle		circle in the plane
date		calendar date (year, month, day)
double precision	float8	double precision floating-point number
inet		IPv4 or IPv6 host address
integer	int, int4	signed four-byte integer
interval [ (p) ]		time span
line		infinite line in the plane
lseg		line segment in the plane
macaddr		MAC address
money		currency amount
numeric [ (p, s) ]	decimal [ (p, s) ]	exact numeric of selectable precision
path		geometric path in the plane
point		geometric point in the plane
polygon		closed geometric path in the plane
real	float4	single precision floating-point number
smallint	int2	signed two-byte integer
serial	serial4	autoincrementing four-byte integer
text		variable-length character string
time [ (p) ] [ without time zone ]		time of day
time [ (p) ] with time zone	timetz	time of day, including time zone
timestamp [ (p) ] [ without time zone ]		date and time
timestamp [ (p) ] with time zone	timestampz	date and time, including time zone

tsquery		text search query
tsvector		text search document
txid_snapshot		user-level transaction ID snapshot
uuid		universally unique identifier
xml		XML data

## Length and Decimals

Use the **Length** edit box to define the length of the field and use **Decimals** edit box to define the number of digits after the decimal point (the scale) for Floating Point data type.

**Note:** Be careful when shortening the field length as losing data might be caused.

### Allow Null

Allow the NULL values for the field.

### Primary Key

A Primary Key is a single field or combination of fields that uniquely defines a record. None of the fields that are part of the primary key can contain a null value.

### Primary Key Name

Right-click and select **Primary Key Name** from the popup menu to enter the primary key constraint name.

### Fill Factor

Right-click and select **Fill Factor** from the popup menu to enter the storage parameter. The fillfactor for a table is a percentage between 10 and 100.

## Setting Other PostgreSQL Table Field Properties

To set the default value for the field use the **Default** edit box.

To set any optional text describing the current field use the **Comment** edit box.

To set the dimensions of array specifiers use the **Dimensions** edit box.

For **Domain** and **Type** data types:

### **Object Schema**

Set the object schema for the field.

### **Object Type**

Set the object type for the field.

## PostgreSQL Table Indexes



Indexes are primarily used to enhance database performance (though inappropriate use can result in slower performance).

An index field can be an expression computed from the values of one or more columns of the table row. This feature can be used to obtain fast access to data based on some transformation of the basic data.

Table indexes are managed on the **Indexes** tab of the Table Designer. Just simply click/double-click an index field for editing. A right-click displays the popup menu or using the index toolbar, allowing you to create new, edit and delete the selected index field.

### Add Index

To add a table index

- Open the table in the Table Designer.
- Open the **Indexes** tab.
- Right-click and select the  **Add Index** from the popup menu or click the  **Add Index** from the toolbar.
- Edit index properties.



### Edit Index

To edit a table index


- Open the table in the Table Designer.
- Open the **Indexes** tab.
- Just simply click/double-click on the index to edit.

### Delete Index


To delete a table index

- Open the table in the Table Designer.
- Open the **Indexes** tab.
- Right-click on the index to delete and select the  **Delete Index** from the popup menu or click the  **Delete Index** from the toolbar.
- Confirm deleting in the dialog window.

## Setting PostgreSQL Table Index Properties

Name	Fields	Index method	Unique	Clustered
▶ cust_index	CustNo 	B-Tree	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Use the **Name** edit box to set the index name. No schema name can be included here; the index is always created in the same schema as its parent table.

To include field(s) in the index, just simply double-click the **Fields** field or click  to open the editor for editing.

**Note:** Some of field types do not allow indexing by several fields.

The **Index method** dropdown list defines the type of the table index. PostgreSQL provides the index methods B-tree, R-tree, hash, and GiST. The B-tree index method is an implementation of Lehman-Yao high-concurrency B-trees. The R-tree index method implements standard R-trees using Guttman's quadratic split algorithm. The hash index method is an implementation of Litwin's linear hashing. Users can also define their own index methods, but that is fairly complicated.

### **Unique**

Makes index unique, causes the system to check for duplicate values in the table when the index is created (if data already exist) and each time data is added.

### **Clustered**

*CLUSTER* instructs PostgreSQL to cluster the table specified by tablename based on the index specified by indexname. The index must already have been defined on tablename.

When a table is clustered, PostgreSQL remembers on which index it was clustered. The form *CLUSTER* tablename reclusters the table on the same index that it was clustered before.

### **Tablespace**

The tablespace in which to create the index.

### **Constraints**

If you wish to create partial index, enter constraint condition in this edit box. A partial index is an index that contains entries for only a portion of a table, usually a portion that is more useful for indexing than the rest of the table.

The **Comment** edit box defines the comment for the index..



## PostgreSQL Table Foreign Keys

A foreign key specifies that the values in a column (or a group of columns) must match the values appearing in some row of another table. We say this maintains the referential integrity between two related tables.

Foreign Keys are managed on the **Foreign Keys** tab of the Table Designer. Just simply click/double-click a foreign key field for editing. A right-click displays the popup menu or using the foreign key toolbar, allowing you to create new, edit and delete the selected foreign key field.

### Add Foreign Key

To add a foreign key

- Open the table in the Table Designer.
- Open the **Foreign Keys** tab.
- Right-click and select the  **Add Foreign Key** from the popup menu or click the  **Add Foreign Key** from the toolbar.
- Edit foreign key properties.



### Edit Foreign Key

To edit a foreign key

- Open the table in the Table Designer.
- Open the **Foreign Keys** tab.
- Just simply click/double-click on the foreign key to edit.

### Delete Foreign Key

To delete a foreign key

- Open the table in the Table Designer.
- Open the **Foreign Keys** tab.
- Right-click on the foreign key to delete and select the  **Delete Foreign Key** from the popup menu or click the  **Delete Foreign Key** from the toolbar.
- Confirm deleting in the dialog window.

## Setting PostgreSQL Table Foreign Key Properties

Name	Fields	Reference Schema	Reference Table	Reference Fields	On Delete	On Update
orders_cust_fk	CustNo  <input type="button" value="..."/>	report	customer	CustNo	NO ACTION	NO ACTION

Use the **Name** edit box to enter a name for the new key and then select a table field to include in the key from the **Fields** group.

Use the **Reference Schema** and **Reference Table** dropdown lists to select a foreign schema and table respectively.

To include field(s) to the key, just simply double-click the **Fields/Reference Fields** field or click  to open the editor(s) for editing.

The **On Delete** and **On Update** dropdown list define the type of the actions to be taken.

### Restrict

Produce an error indicating that the deletion or update would create a foreign key constraint violation. This is the same as NO ACTION except that the check is not deferrable.

### No Action

Produce an error indicating that the deletion or update would create a foreign key constraint violation. If the constraint is deferred, this error will be produced at constraint check time if there still exist any referencing rows. This is the default action.

### Cascade

Delete any rows referencing the deleted row, or update the value of the referencing column to the new value of the referenced column, respectively.

### Set Null

Set the referencing column(s) to null.

### Set Default

Set the referencing column(s) to their default values.



## PostgreSQL Table Uniques

Unique constraints ensure that the data contained in a column or a group of columns is unique with respect to all the rows in the table.

Uniques are managed on the **Uniques** tab of the Table Designer. Just simply click/double-click an unique field for editing. Using the unique toolbar, allowing you to create new, edit and delete the selected unique field.

### Add Unique

To add an unique

- Open the table in the Table Designer.
- Open the **Uniques** tab.
- Right-click and select the  **Add Unique** from the popup menu or click the  **Add Unique** from the toolbar.
- Edit unique properties.



### Edit Unique

To edit an unique


- Open the table in the Table Designer.
- Open the **Uniques** tab.
- Just simply click on the unique to edit.

### Delete Unique

To delete an unique


- Open the table in the Table Designer.
- Open the **Uniques** tab.
- Right-click on the unique to delete and select the  **Delete Unique** from the popup menu or click the  **Delete Unique** from the toolbar.
- Confirm deleting in the dialog window.

## Setting PostgreSQL Table Unique Properties

Name	Fields
▶ order_cust_index	OrderNo, CustNo 

Use the **Name** edit box to set the unique name.

### Fields

To set field(s) as unique, just simply double-click the **Fields** field or click  to open the editor(s) for editing.

Select the field(s) from the list. To remove the fields from the unique, uncheck them in the same way.

### Tablespace

Allows setting a tablespace different from the default tablespace.

The **Comment** edit box defines the comment for the unique.

### Fill Factor

The fillfactor for a unique is a percentage between 10 and 100. 100 (complete packing) is the default.

**Note:** Support from PostgreSQL 8.2 or later.



## PostgreSQL Table Checks

A check constraint is the most generic constraint type. It allows you to specify that the value in a certain column must satisfy a Boolean (truth-value) expression.

Checks are managed on the **Checks** tab of the Table Designer. Just simply click/double-click a check field for editing. Using the check toolbar, allowing you to create new, edit and delete the selected check field.

### Add Check

To add a check

- Open the table in the Table Designer.
- Open the **Checks** tab.
- Right-click and select the  **Add Check** from the popup menu or click the  **Add Check** from the toolbar.
- Edit check properties.



### Edit Check

To edit a check

- Open the table in the Table Designer.
- Open the **Checks** tab.
- Just simply click on the check to edit.

### Delete Check

To delete a check

- Open the table in the Table Designer.
- Open the **Checks** tab.
- Right-click on the check to delete and select the  **Delete Check** from the popup menu or click the  **Delete Check** from the toolbar.
- Confirm deleting in the dialog window.

## Setting PostgreSQL Table Check Properties

Use the **Name** edit box to set the check name.

### Check

Set the condition for checking, e.g. "field\_name1 > 0 AND field\_name2 > field\_name1" in the **Check** edit box. A check constraint specified as a column constraint should reference that column's value only, while an expression appearing in a table constraint may reference multiple columns.

### Definition

allows you to enter the definition for the check.

### Comment

allows you to enter the comment for the check.

## PostgreSQL Table Excludes



A exclude constraint guarantees that if any two rows are compared on the specified column(s) or expression(s) using the specified operator(s), not all of these comparisons will return TRUE.

Excludes are managed on the **Excludes** tab of the Table Designer. Just simply click/double-click an exclude field for editing. Using the exclude toolbar, allowing you to create new, edit and delete the selected exclude field.

**Note:** Exclude is supported from PostgreSQL 9.0 or later.

### Add Exclude

To add an exclude

- Open the table in the Table Designer.
- Open the **Excludes** tab.
- Right-click and select the  **Add Exclude** from the popup menu or click the  **Add Exclude** from the toolbar.
- Edit exclude properties.



### Edit Exclude

To edit an exclude

- Open the table in the Table Designer.
- Open the **Excludes** tab.
- Just simply click on the exclude to edit.

### Delete Exclude

To delete an exclude

- Open the table in the Table Designer.
- Open the **Excludes** tab.
- Right-click on the exclude to delete and select the  **Delete Exclude** from the popup menu or click the  **Delete Exclude** from the toolbar.
- Confirm deleting in the dialog window.

## Setting PostgreSQL Table Exclude Properties

Use the **Name** edit box to set the exclude name.

### **Index method**

The name of the index access method to be used.

### **Element**

Choose the element(s) to be excluded and specify the operator(s).

### **Tablespace**

The tablespace in which to create the index.

### **Fill Factor**

The fillfactor for an index is a percentage that determines how full the index method will try to pack index pages.

### **Predicate**

Specify an exclusion constraint on a subset of the table.

The **Comment** edit box defines the comment for the exclude.

## PostgreSQL Table Rules



The PostgreSQL rule system allows one to define an alternate action to be performed on insertions, updates, or deletions in database tables. Roughly speaking, a rule causes additional commands to be executed when a given command on a given table is executed.

**Note:** You must be the owner of a table to create or change rules for it.

Rules are managed on the **Rules** tab of the Table Designer. Just simply click/double-click a rule field for editing. Using the rule toolbar, allowing you to create new, edit and delete the selected rule field.

### Add Rule

To add a rule

- Open the table in the Table Designer.
- Open the **Rules** tab.
- Right-click and select the  **Add Rule** from the popup menu or click the  **Add Rule** from the toolbar.
- Edit rule properties.



### Edit Rule

To edit a rule

- Open the table in the Table Designer.
- Open the **Rules** tab.
- Just simply click on the rule to edit.

### Delete Rule

To delete a rule

- Open the table in the Table Designer.
- Open the **Rules** tab.
- Right-click on the rule to delete and select the  **Delete Rule** from the popup menu or click the  **Delete Rule** from the toolbar.
- Confirm deleting in the dialog window.

## Setting PostgreSQL Table Rule Properties

Use the **Name** edit box to set the rule name. This must be distinct from the name of any other rule for the same table. Multiple rules on the same table and same event type are applied in alphabetical name order.

### Event

The event is one of *SELECT*, *INSERT*, *UPDATE*, or *DELETE*.

### Do instead

This indicates that the commands should be executed instead of the original command. Otherwise, the commands should be executed in addition to the original command.

### Condition

Any SQL conditional expression (returning boolean). The condition expression may not refer to any tables except *NEW* and *OLD*, and may not contain aggregate functions.

### Definition

The command or commands that make up the rule action. Valid commands are *SELECT*, *INSERT*, *UPDATE*, *DELETE*, or *NOTIFY*.

Within condition and command, the special table names *NEW* and *OLD* may be used to refer to values in the referenced table. *NEW* is valid in *ON INSERT* and *ON UPDATE* rules to refer to the new row being inserted or updated. *OLD* is valid in *ON UPDATE* and *ON DELETE* rules to refer to the existing row being updated or deleted

The **Comment** edit box defines the comment for the rule.

## PostgreSQL Table Triggers



A trigger is a specification that the database should automatically execute a particular function whenever a certain type of operation is performed. Triggers can be defined to execute either before or after any INSERT, UPDATE, or DELETE operation, either once per modified row, or once per SQL statement.

Triggers are managed on the **Triggers** tab of the Table Designer. Just simply click a trigger field for editing. A right-click displays the popup menu or using the trigger toolbar, allowing you to create new, edit and delete the selected trigger field.

**Note:** To create a trigger on a table, the user must have the TRIGGER privilege on the table.

### Add Trigger

To add a trigger

- Open the table in the Table Designer.
- Open the **Triggers** tab.
- Right-click and select the  **Add Trigger** from the popup menu or click the  **Add Trigger** from the toolbar.
- Edit trigger properties.



### Edit Trigger

To edit a trigger

- Open the table in the Table Designer.
- Open the **Triggers** tab.
- Just simply click on the trigger to edit.

### Delete Trigger

To delete a trigger

- Open the table in the Table Designer.
- Open the **Triggers** tab.
- Right-click on the trigger to delete and select the  **Delete Trigger** from the popup menu or click the  **Delete Trigger** from the toolbar.
- Confirm deleting in the dialog window.

## Setting PostgreSQL Table Trigger Properties

Use the **Name** edit box to set the trigger name. This must be distinct from the name of any other trigger for the same table.

### Row trigger

This specifies whether the trigger procedure should be fired once for every row affected by the trigger event, or just once per SQL statement. If unchecked, *FOR EACH STATEMENT* is the default.

Use the **Fires** dropdown list to define the trigger action time. It can be **Before** or **After** to indicate that the trigger activates before or after the statement that activated it.

### Insert

The trigger is activated whenever a new row is inserted into the table.

### Update

The trigger is activated whenever a row is modified.

### Delete

The trigger is activated whenever a row is deleted from the table.

### Update Of Fields

Specify a list of columns. The trigger will only fire if at least one of the listed columns is mentioned as a target of the UPDATE command.

**Note:** Support from PostgreSQL 9.1 or later.

### When Clause

Specify a Boolean WHEN condition, which will be tested to see whether the trigger should be fired.

**Note:** Support from PostgreSQL 9.0 or later.

### Trigger Function Schema and Trigger Function

A user-supplied function that is declared as taking no arguments and returning type trigger, which is executed when the trigger fires.

## **Arguments**

An optional comma-separated list of arguments to be provided to the function when the trigger is executed. The arguments are literal string constants. Simple names and numeric constants may be written here, too, but they will all be converted to strings. Please check the description of the implementation language of the trigger function about how the trigger arguments are accessible within the function; it may be different from normal function arguments.

The **Comment** edit box defines the comment for the trigger.

## PostgreSQL Table Options


The **Owner** drop-down list defines the user to own this table.

The **Tablespace** drop-down list defines a tablespace different from the default tablespace to create a table.

**Note:** Support from PostgreSQL 8.0 or later.

### Inherits from

This option specifies a list of tables from which the new table automatically inherits all columns. Use of inheritance creates a persistent relationship between the new child table and its parent table(s). Schema modifications to the parent(s) normally propagate to children as well, and by default the data of the child table is included in scans of the parent(s).

To set the new table to be inherited from one or several existing tables, just simply click  to open the editor(s) for editing.

### Has Oids

Check this option if you want to specify whether rows of the new table should have OIDs (object identifiers) assigned to them.


### Fill Factor

The fillfactor for a table is a percentage between 10 and 100. 100 (complete packing) is the default. When a smaller fillfactor is specified, INSERT operations pack table pages only to the indicated percentage; the remaining space on each page is reserved for updating rows on that page. This gives UPDATE a chance to place the updated copy of a row on the same page as the original, which is more efficient than placing it on a different page. For a table whose entries are never updated, complete packing is the best choice, but in heavily updated tables smaller fillfactors are appropriate.

**Note:** Support from PostgreSQL 8.2 or later.



## PostgreSQL Views


Views are useful for allowing users to access a set of relations (tables) as if it were a single table, and limiting their access to just that. Views can also be used to restrict access to rows (a subset of a particular table).

Just simply click  to open an object pane for **View**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit, open and delete the selected view.

### Create View

To create a new view

- Select anywhere on the object pane.
- Click the  **New View** from the object pane toolbar.  
or
- Right-click and select  **New View** from the popup menu.
- Edit view properties on the appropriate tabs of the View Designer.

**Hint:** To create new view you can also right-click the Views node of the navigation pane and select the  **New View** from the popup menu.

To create a new view with the same properties as one of the existing views has (using drag and drop method)




**Apply to:** current database {same connection}

- Select the view(s) for copying in the navigation pane/object pane.
- Right-click and drag the chosen view(s) to the target location.
- Select one of the following options:
  - Copy here (Structure and Data)
  - Copy here (Structure only)
  - Move here
  - Cancel
- The newly created view(s) will be named as "viewname\_**copy**".




**Apply to:** different database {same connection}  
different database {different connection} (Data Transfer tool will be activated)

- Select the view(s) for copying in the object pane.
- Drag and drop the chosen view(s) to the target database.
- Select one of the following options:
  - Copy here (Structure and Data)
  - Copy here (Structure only)
  - Cancel

To create a new view with modification as one of the existing views

- Select the view for modifying in the navigation pane/object pane.
- Right-click and select the  **Design View** from the popup menu.  
or
- Click the  **Design View** from the object pane toolbar.
- Modify view properties on the appropriate tabs of the View Designer.
- Click  **Save As**.

To create a new view with loading from a SQL file

- Select anywhere on the object pane.
- Click the  **New View** from the object pane toolbar.  
or
- Right-click and select  **New View** from the popup menu.
- Click  **Load**.

## Create View Shortcut



To create a view shortcut

- Select the view for editing in the navigation pane/object pane.
- Right-click and select **Create Open View Shortcut...** from the popup menu.
- Define the location you wish your shortcut to be saved.

**Note:** This option is used to provide a convenient way for you to open your view for entering data directly (Grid View/Form View) without activating the main Navicat.

## Edit View

To edit the existing view (manage its SQL definition etc)



- Select the view for editing in the navigation pane/object pane.
- Right-click and select the  **Design View** from the popup menu.  
or
- Click the  **Design View** from the object pane toolbar.
- Edit view properties on the appropriate tabs of the View Designer.

To change the name of the view

- Select the view for editing in the navigation pane/object pane.
- Right-click and select the **Rename** from the popup menu.



## Open View

To open a view (manage view data)

- Select the view for opening in the navigation pane/object pane.
- Right-click and select the  **Open View** from the popup menu or simply double-click the view.  
or
- Click the  **Open View** from the object pane toolbar.

## Delete View

To delete a view

- Select the view for deleting in the navigation pane/object pane.
- Right-click and select the  **Delete View** from the popup menu.  
or
- Click the  **Delete View** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve View Information

To achieve a view information

- Select the view in the object pane.
- Right-click the selected view and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL View Designer

**View Designer** is the basic Navicat tool for working with views. It allows you to create new view and edit the existing view definition (view name and the SELECT statement it implements).

- [Working with View Builder](#)
- [Editing View SQL Definition](#)
- [Setting Advanced View Properties](#)
- Editing View Comment
- View SQL Preview
- [View Preview](#)
- [View Explain](#)

## **Working with PostgreSQL View Builder (Available only in Full Version)**

**View Builder** allows you to build views visually. It allows you to create and edit views without knowledge of SQL. See Query Builder for details.

## Editing PostgreSQL View SQL Definition

The **Definition** tab allows you to edit the view definition as SQL statement (SELECT statement it implements).

Example:

```
SELECT
    report.clients.RecordID
FROM
    report.clients
```


**Hint:** To customize the view of the editor and find out more features for sql editing, see [Editor View and More Features](#).

## Setting Advanced PostgreSQL View Properties

### **Owner**


The owner of the view.

## PostgreSQL View Preview

To preview the result of the view, click  **Preview** on the toolbar. If the query statement is correct, the **Result** and **Message** tabs will be opened.

The **Result** tab displays the data of the view as a grid and the **Message** tab displays the message log.

## PostgreSQL View Explain

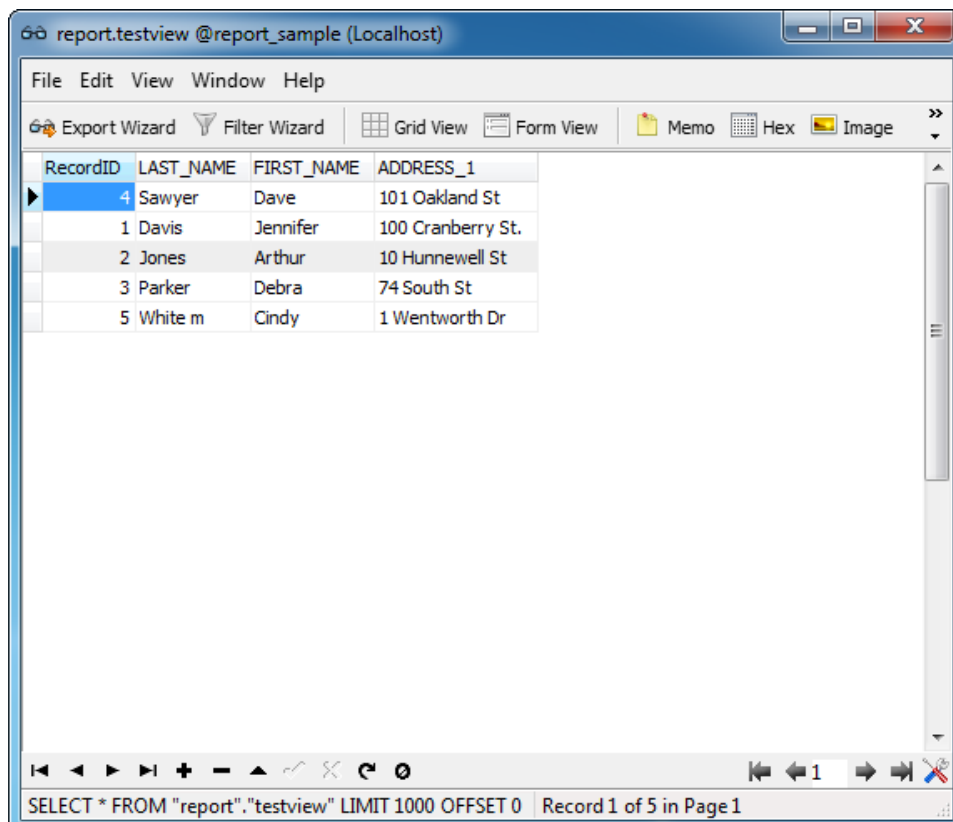
To show the Query Plan of the view, click  **Explain** on the toolbar. If the query statement is correct, the **Message** tab will show the query plan.

## PostgreSQL View Viewer

**View Viewer** displays the view data as a grid. Data can be displayed in three modes:  **Grid View**,  **Form View** and **Text/Blob View**. See Data View for details.

The toolbars of View Viewer provides the following functions for managing data:

- **Export Data**  
Export data to MS Word, MS Excel, MS Access, TXT, DBF, HTML, SQL, RTF and more.
- **Filter Data**  
Allow you to filter records by creating and applying filter criteria for the data grid.
- **Edit TEXT/BLOB**  
Allow you to view and edit the content of TEXT and BLOB fields.




## PostgreSQL Functions

PostgreSQL provides four kinds of functions:



- query language functions (functions written in SQL)
- procedural language functions (functions written in, for example, PL/Tcl or PL/pgSQL)
- internal functions
- C-language functions


Every kind of function can take base types, composite types, or combinations of these as arguments (parameters). In addition, every kind of function can return a base type or a composite type. Many kinds of functions can take or return certain pseudo-types (such as polymorphic types), but the available facilities vary.

Just simply click  to open an object pane for **Function**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected function.

### Create Function

To create a new function

- Select anywhere on the object pane.
- Click the  **New Function** from the object pane toolbar.  
or
- Right-click and select  **New Function** from the popup menu.
- Edit function properties on the appropriate tabs of the Function Designer.

**Hint:** To create new function you can also right-click the Function node of the navigation pane and select the  **New Function** from the popup menu.

To create a new function with the same properties as one of the existing function has (using drag and drop method)

**Apply to:** current database {same connection}




- Select the function(s) for copying in the navigation pane/object pane.
- Right-click and drag the chosen function(s) to the target location.
- Select one of the following options:
  - Copy here (Structure and Data)
  - Copy here (Structure only)
  - Move here
  - Cancel
- The newly created function(s) will be named as "functionname\_**copy**".

**Apply to:** different database {same connection}

different database {different connection} (Data Transfer tool will be activated)



- Select the function(s) for copying in the object pane.
- Drag and drop the chosen function(s) to the target database.
- Select one of the following options:
  - Copy here (Structure and Data)
  - Copy here (Structure only)
  - Cancel

To create a new function with modification as one of the existing function

- Select the function for modifying in the navigation pane/object pane.
- Right-click and select the  **Design Function** from the popup menu or simply double-click the function.  
or
- Click the  **Design Function** from the object pane toolbar.
- Modify function properties on the appropriate tabs of the Function Designer.
- Click  **Save As**.

## Edit Function

To edit the existing function (manage its definition, advanced etc)



- Select the function for editing in the navigation pane/object pane.
- Right-click and select the  **Design Function** from the popup menu or simply double-click the function.  
or
- Click the  **Design Function** from the object pane toolbar.
- Edit function properties on the appropriate tabs of the Function Designer.

To change the name of the function


- Select the function for editing in the navigation pane/object pane.
- Right-click and select the **Rename** from the popup menu.

## Run Function

To run a function in the navigation pane/object pane



- Select the function for executing in the navigation pane/object pane.
- Click the  **Execute Function** from the object pane toolbar.  
or
- Right-click and select  **Execute Function** from the popup menu.
- View/edit the returned data on the Result tab.

To run a function in the Function Designer

- Create a new function/open the existing function.
- Click  **Run**.
- View/edit the returned data on the Result tab.

## Delete Function

To delete a function


- Select the function for deleting in the navigation pane/object pane.
- Right-click and select the  **Delete Function** from the popup menu.  
or
- Click the  **Delete Function** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Function Information

To achieve a function information

- Select the function in the object pane.
- Right-click the selected function and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Function Wizard

Click the  **New Function** from the object pane toolbar. The **Function Wizard** will pop up and it allows you to create a function easily.

- [Setting Parameters for Function](#)
- [Setting Return Type for Function](#)

You are allowed not to show the **Function Wizard** when create new function.

**Hint:** Once uncheck the **Show wizard next time**, you can go to Options to enable it.

## Setting Parameters for PostgreSQL Function

### Function

Define the parameter(s) of the function. Set the parameter **Mode**, **Type Schema**, **Type**, **Name** and **Default Value** under corresponding columns.

## Setting Return Type for PostgreSQL Function

Select the **Schema** and **Return Type** from the list.

## PostgreSQL Function Designer

**Function Designer** is the basic Navicat tool for working with functions. It allows you to create new function and edit the existing function definition.

- [Editing Function Definition](#)
- [Setting Advanced Function Properties](#)
- Editing Function Comment
- Function SQL Preview
- [Viewing Function Result](#)

## Editing PostgreSQL Function Definition

Edit the function definition under the **Definition** tab. Definition consists of a valid SQL procedure statement. This can be a simple statement such as *SELECT* or *INSERT*, or it can be a compound statement written using *BEGIN* and *END*. Compound statements can contain declarations, loops, and other control structure statements. The general form of these statements follows.

Example:

```
BEGIN
  RETURN i + j;
END
```

**Hint:** To customize the view of the editor and find out more features for sql editing, see Editor View and More Features.

### **Parameter**

Defines function parameter.

### **Return type schema and Return Type**

It indicates the return type of the function.

## Setting Advanced PostgreSQL Function Properties

### Owner

The owner of the function.

**Note:** Support from PostgreSQL 8.0 or later.

### Language

The name of the language that the function is implemented in. May be SQL, C, internal, or the name of a user-defined procedural language. For backward compatibility, the name may be enclosed by single quotes.

### Volatility

These attributes inform the query optimizer about the behavior of the function. At most one choice may be specified. If none of these appear, VOLATILE is the default assumption.

**IMMUTABLE** indicates that the function cannot modify the database and always returns the same result when given the same argument values; that is, it does not do database lookups or otherwise use information not directly present in its argument list. If this option is given, any call of the function with all-constant arguments can be immediately replaced with the function value.

**STABLE** indicates that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same argument values, but that its result could change across SQL statements. This is the appropriate selection for functions whose results depend on database lookups, parameter variables (such as the current time zone), etc. Also note that the `current_timestamp` family of functions qualify as stable, since their values do not change within a transaction.

**VOLATILE** indicates that the function value can change even within a single table scan, so no optimizations can be made. Relatively few database functions are volatile in this sense; some examples are `random()`, `currval()`, `timeofday()`. But note that any function that has side-effects must be classified volatile, even if its result is quite predictable, to prevent calls from being optimized away; an example is `setval()`.

### Security of definer

Specifies that the function is to be executed with the privileges of the user that created it.

### Returns Set

Indicates that the function will return a set of items, rather than a single item.

## **Strict**

Indicates that the function always returns null whenever any of its arguments are null. If this parameter is specified, the function is not executed when there are null arguments; instead a null result is assumed automatically.

## **Estimated cost**

A positive number giving the estimated execution cost for the function, in units of `cpu_operator_cost`. If the function returns a set, this is the cost per returned row. If the cost is not specified, 1 unit is assumed for C-language and internal functions, and 100 units for functions in all other languages. Larger values cause the planner to try to avoid evaluating the function more often than necessary.

**Note:** Support from PostgreSQL 8.3 or later.

## **Estimated rows**

A positive number giving the estimated number of rows that the planner should expect the function to return. This is only allowed when the function is declared to return a set. The default assumption is 1000 rows.


**Note:** Support from PostgreSQL 8.3 or later.

## **Configuration parameter**

The specified configuration parameter to be set to the specified value when the function is entered, and then restored to its prior value when the function exits.

**Note:** Support from PostgreSQL 8.3 or later.

## Viewing PostgreSQL Function Result

To run the function click  **Run** on the toolbar. If the SQL statement is correct, the statement will be executed and, if the statement is supposed to return data, the **Result** tab opens with the data returned by the function. If an error occurs while executing the function, execution stops, the appropriate error message is displayed.


If the function requires input parameter, the **Input Parameters** box will popup. Use ',' to separate the parameters.

**Note:** The **Result** tab displays the result data as grid.

**Hint:** Navicat supports to return 10 resultsets.



## PostgreSQL Aggregates

Aggregate functions in PostgreSQL are expressed as state values and state transition functions. That is, an aggregate can be defined in terms of state that is modified whenever an input item is processed. To define a new aggregate function, one selects a data type for the state value, an initial value for the state, and a state transition function. The state transition function is just an ordinary function that could also be used outside the context of the aggregate. A final function can also be specified, in case the desired result of the aggregate is different from the data that needs to be kept in the running state value.




Just simply click -> Aggregate to open an object pane for **Aggregate**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected aggregate.

### Create Aggregate

To create a new aggregate



- Select anywhere on the object pane.
- Click the  **New Aggregate** from the object pane toolbar.  
or
- Right-click and select  **New Aggregate** from the popup menu.
- Edit aggregate properties on the appropriate tabs of the Aggregate Designer.

To create a new aggregate with modification as one of the existing aggregate

- Select the aggregate for modifying in the object pane.
- Right-click and select the  **Design Aggregate** from the popup menu or simply double-click the aggregate.  
or
- Click the  **Design Aggregate** from the object pane toolbar.
- Modify aggregate properties on the appropriate tabs of the Aggregate Designer.
- Click  **Save As**.

## Edit Aggregate

To edit the existing aggregate (manage its properties etc)

- Select the aggregate for editing in the object pane.
- Right-click and select the  **Design Aggregate** from the popup menu or simply double-click the aggregate.  
or
- Click the  **Design Aggregate** from the object pane toolbar.
- Edit aggregate properties on the appropriate tabs of the Aggregate Designer.



To change the name of the aggregate

- Select the aggregate for editing in the object pane.
- Right-click and select the **Rename** from the popup menu.

**Note:** Support from PostgreSQL 7.4 or later.

## Delete Aggregate

To delete an aggregate

- Select the aggregate for deleting in the object pane.
- Right-click and select the  **Delete Aggregate** from the popup menu.  
or
- Click the  **Delete Aggregate** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Aggregate Information

To achieve an aggregate information

- Select the aggregate in the object pane.
- Right-click the selected aggregate and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Aggregate Designer

**Aggregate Designer** is the basic Navicat tool for working with aggregates. It allows you to create new aggregate and edit the existing aggregate properties.

- [Editing Aggregate Properties](#)
- Editing Aggregate Comment
- Aggregate SQL Preview

## Editing PostgreSQL Aggregate Properties

### Owner

The owner of the aggregate function.

**Note:** Support from PostgreSQL 8.0 or later.

### Input type

An input data type on which this aggregate function operates.

**Note:** Support from PostgreSQL 8.2 or later. For versions below 8.2, just select the **Input type schema** and **Input type** from the dropdown lists.

### State type schema and State type

The data type for the aggregate's state value.

### State function schema and State function

The state transition function to be called for each input row. For an N-argument aggregate function, the state function must take N+1 arguments, the first being of type *state\_data\_type* and the rest matching the declared input data type(s) of the aggregate. The function must return a value of type *state\_data\_type*. This function takes the current state value and the current input data value(s), and returns the next state value.

### Final function schema and Final function

The final function called to compute the aggregate's result after all input rows have been traversed. The function must take a single argument of type *state\_data\_type*. The return data type of the aggregate is defined as the return type of this function. If final function is not specified, then the ending state value is used as the aggregate's result, and the return type is *state\_data\_type*.

### Initial condition

The initial setting for the state value. This must be a string constant in the form accepted for the data type *state\_data\_type*. If not specified, the state value starts out null.


### Sort operator schema and Sort operator

The associated sort operator for a MIN- or MAX-like aggregate. The operator is assumed to have the same input data types as the aggregate (which must be a single-argument aggregate).

**Note:** Support from PostgreSQL 8.1 or later.



## PostgreSQL Conversions

Conversion defines a new conversion between character set encodings. Conversion names may be used in the convert function to specify a particular encoding conversion. Also, conversions that are marked DEFAULT can be used for automatic encoding conversion between client and server. For this purpose, two conversions, from encoding A to B and from encoding B to A, must be defined.




Just simply click -> Conversion to open an object pane for **Conversion**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected conversion.

### Create Conversion

To create a new conversion



- Select anywhere on the object pane.
- Click the  **New Conversion** from the object pane toolbar.  
or
- Right-click and select  **New Conversion** from the popup menu.
- Edit conversion properties on the appropriate tabs of the Conversion Designer.

To create a new conversion with modification as one of the existing conversion

- Select the conversion for modifying in the object pane.
- Right-click and select the  **Design Conversion** from the popup menu or simply double-click the conversion.  
or
- Click the  **Design Conversion** from the object pane toolbar.
- Modify conversion properties on the appropriate tabs of the Conversion Designer.
- Click  **Save As**.

## Edit Conversion

To edit the existing conversion (manage its properties etc)

- Select the conversion for editing in the object pane.
- Right-click and select the  **Design Conversion** from the popup menu or simply double-click the conversion.  
or
- Click the  **Design Conversion** from the object pane toolbar.
- Edit conversion properties on the appropriate tabs of the Conversion Designer.



To change the name of the conversion

- Select the conversion for editing in the object pane.
- Right-click and select the **Rename** from the popup menu.

**Note:** Support from PostgreSQL 7.4 or later.

## Delete Conversion

To delete a conversion

- Select the conversion for deleting in the object pane.
- Right-click and select the  **Delete Conversion** from the popup menu.  
or
- Click the  **Delete Conversion** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Conversion Information

To achieve a conversion information

- Select the conversion in the object pane.
- Right-click the selected conversion and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Conversion Designer

**Conversion Designer** is the basic Navicat tool for working with conversions. It allows you to create new conversion and edit the existing conversion properties.

- [Editing Conversion Properties](#)
- Editing Conversion Comment
- Conversion SQL Preview

## Editing PostgreSQL Conversion Properties

### Owner

The owner of the conversion function.

**Note:** Support from PostgreSQL 8.0 or later.

### Source encoding

The source encoding name.

### Target encoding

The destination encoding name.

### Schema of function and Function

The function used to perform the conversion. The function name may be schema-qualified. If it is not, the function will be looked up in the path.

The function must have the following signature:

```
conv_proc(
integer, -- source encoding ID
integer, -- destination encoding ID
cstring, -- source string (null terminated C string)
internal, -- destination (fill with a null terminated C string)
integer -- source string length
) RETURNS void;
```


### Default

Check this box to indicate that this conversion is the default for this particular source to destination encoding. There should be only one default encoding in a schema for the encoding pair.

## PostgreSQL Domains



A domain is essentially a data type with optional constraints (restrictions on the allowed set of values). The user who defines a domain becomes its owner.

Domains are useful for abstracting common constraints on fields into a single location for maintenance. For example, several tables might contain email address columns, all requiring the same *CHECK* constraint to verify the address syntax. Define a domain rather than setting up each table's constraint individually.




Just simply click  -> Domain to open an object pane for **Domain**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected domain.

### Create Domain

To create a new domain



- Select anywhere on the object pane.
- Click the  **New Domain** from the object pane toolbar.  
or
- Right-click and select  **New Domain** from the popup menu.
- Edit domain properties on the appropriate tabs of the Domain Designer.

To create a new domain with modification as one of the existing domain

- Select the domain for modifying in the object pane.
- Right-click and select the  **Design Domain** from the popup menu or simply double-click the domain.  
or
- Click the  **Design Domain** from the object pane toolbar.
- Modify domain properties on the appropriate tabs of the Domain Designer.
- Click  **Save As**.

## Edit Domain

To edit the existing domain (manage its general etc)



- Select the domain for editing in the object pane.
- Right-click and select the  **Design Domain** from the popup menu or simply double-click the domain.  
or
- Click the  **Design Domain** from the object pane toolbar.
- Edit domain properties on the appropriate tabs of the Domain Designer.

To change the name of the domain

- Select the domain for editing in the object pane.
- Right-click and select the **Rename** from the popup menu.

## Delete Domain

To delete a domain

- Select the domain for deleting in the object pane.
- Right-click and select the  **Delete Domain** from the popup menu.  
or
- Click the  **Delete Domain** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Domain Information

To achieve a domain information

- Select the domain in the object pane.
- Right-click the selected domain and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Domain Designer

**Domain Designer** allows you to define domain properties and its checks. It allows you to create new domain and edit the existing domain properties.

- [Editing Domain General](#)
- [Editing Domain Check](#)
- Editing Domain Comment
- Domain SQL Preview

## Editing PostgreSQL Domain General

### Underlying Type Category

Choose the underlying data type category: **Base Type**, **Composite Type**, **Enum Type** and **Domain**.

**Note:** Support from PostgreSQL 8.2 or later.

### Underlying Type Schema

Select schema of the underlying data type.

### Underlying Type

Select the underlying data type of the domain from the drop-down list.

### Dimensions

The dimensions of array specifiers.

### Length and Scale

Use the **Length** edit box to define the length of the field and use **Scale** edit box to define the number of digits after the decimal point. (if required for the selected data type)

### Default

The *DEFAULT* clause specifies a default value for columns of the domain data type. The value is any variable-free expression (but subqueries are not allowed). The data type of the default expression must match the data type of the domain. If no default value is specified, then the default value is the null value.

The default expression will be used in any insert operation that does not specify a value for the column. If a default value is defined for a particular column, it overrides any default associated with the domain. In turn, the domain default overrides any default value associated with the underlying data type.

### Not null

Values of this domain are not allowed to be null.

### Owner

The owner of the domain function. The user who defines a domain becomes its owner.

**Note:** Support from PostgreSQL 7.4 or later.

## Editing PostgreSQL Domain Check

The **Checks** tab is provided for managing domain checks. It allows you to create new, edit, or delete the selected check.

*CHECK* clauses specify integrity constraints or tests which values of the domain must satisfy. Each constraint must be an expression producing a Boolean result. It should use the key word *VALUE* to refer to the value being tested.


See [Checks](#) for details.

## PostgreSQL Trigger Functions

Trigger Function can be created with PL/pgSQL and referenced within a PostgreSQL trigger definition. The term "trigger function" is simply a way of referring to a function that is intended to be invoked by a trigger. Triggers define operations that are performed when a specific event occurs within the database. A PL/pgSQL trigger function can be referenced by a trigger as the operation to be performed when the trigger's event occurs.



The definition of a trigger and the definition of its associated trigger function are two different things. A trigger is defined with the SQL *CREATE TRIGGER* command, whereas trigger functions are defined using the SQL *CREATE FUNCTION* command.

See [Triggers](#) for details.




Just simply click  -> Trigger Function to open an object pane for **Trigger Function**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected trigger function.

### Create Trigger Function

To create a new trigger function



- Select anywhere on the object pane.
- Click the  **New Trigger Function** from the object pane toolbar.  
or
- Right-click and select  **New Trigger Function** from the popup menu.
- Edit trigger function properties on the appropriate tabs of the Trigger Function Designer.

To create a new trigger function with modification as one of the existing trigger function

- Select the trigger function for modifying in the object pane.
- Right-click and select the  **Design Trigger Function** from the popup menu or simply double-click the trigger function.  
or
- Click the  **Design Trigger Function** from the object pane toolbar.
- Modify trigger function properties on the appropriate tabs of the Trigger Function Designer.
- Click  **Save As**.

## Edit Trigger Function

To edit the existing trigger function (manage its definition, advanced, etc)



- Select the trigger function for editing in the object pane.
- Right-click and select the  **Design Trigger Function** from the popup menu or simply double-click the trigger function.  
or
- Click the  **Design Trigger Function** from the object pane toolbar.
- Edit trigger function properties on the appropriate tabs of the Trigger Function Designer.

To change the name of the trigger function

- Select the trigger function for editing in the object pane.
- Right-click and select the **Rename** from the popup menu.

## Delete Trigger Function

To delete a trigger function

- Select the trigger function for deleting in the object pane.
- Right-click and select the  **Delete Trigger Function** from the popup menu.  
or
- Click the  **Delete Trigger Function** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Trigger Function Information

To achieve a trigger function information

- Select the trigger function in the object pane.
- Right-click the selected trigger function and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Trigger Function Designer

**Trigger Function Designer** is the basic Navicat tool for working with trigger functions. It allows you to create new trigger function and edit the existing trigger function definition.

- [Editing Trigger Function Definition](#)
- [Setting Advanced Trigger Function Properties](#)
- Editing Trigger Function Comment
- Trigger Function SQL Preview

## Editing PostgreSQL Trigger Function Definition

Edit the trigger function definition under the **Definition** tab. Definition consists of a valid SQL procedure statement. This can be a simple statement such as *SELECT* or *INSERT*, or it can be a compound statement written using *BEGIN* and *END*. Compound statements can contain declarations, loops, and other control structure statements.

### **Parameter**

Defines trigger function parameter.

### **Return type schema and Return Type**

It indicates the return type of the trigger function.

**Hint:** To customize the view of the editor and find out more features for sql editing, see Editor View and More Features.

## Setting Advanced PostgreSQL Trigger Function Properties

### Owner

The owner of the trigger function.

**Note:** Support from PostgreSQL 8.0 or later.

### Language

The name of the language that the function is implemented in. May be C, internal, or the name of a user-defined procedural language. For backward compatibility, the name may be enclosed by single quotes.

### Volatility

These attributes inform the query optimizer about the behavior of the function. At most one choice may be specified. If none of these appear, VOLATILE is the default assumption.

**IMMUTABLE** indicates that the function cannot modify the database and always returns the same result when given the same argument values; that is, it does not do database lookups or otherwise use information not directly present in its argument list. If this option is given, any call of the function with all-constant arguments can be immediately replaced with the function value.

**STABLE** indicates that the function cannot modify the database, and that within a single table scan it will consistently return the same result for the same argument values, but that its result could change across SQL statements. This is the appropriate selection for functions whose results depend on database lookups, parameter variables (such as the current time zone), etc. Also note that the `current_timestamp` family of functions qualify as stable, since their values do not change within a transaction.

**VOLATILE** indicates that the function value can change even within a single table scan, so no optimizations can be made. Relatively few database functions are volatile in this sense; some examples are `random()`, `currval()`, `timeofday()`. But note that any function that has side-effects must be classified volatile, even if its result is quite predictable, to prevent calls from being optimized away; an example is `setval()`.

### Security of definer

Specifies that the function is to be executed with the privileges of the user that created it.

### Returns Set

Indicates that the function will return a set of items, rather than a single item.

## **Strict**

Indicates that the function always returns null whenever any of its arguments are null. If this parameter is specified, the function is not executed when there are null arguments; instead a null result is assumed automatically.

## **Estimated cost**

A positive number giving the estimated execution cost for the function, in units of `cpu_operator_cost`. If the function returns a set, this is the cost per returned row. If the cost is not specified, 1 unit is assumed for C-language and internal functions, and 100 units for functions in all other languages. Larger values cause the planner to try to avoid evaluating the function more often than necessary.

**Note:** Support from PostgreSQL 8.2 or later.

## **Estimated rows**

A positive number giving the estimated number of rows that the planner should expect the function to return. This is only allowed when the function is declared to return a set. The default assumption is 1000 rows.

**Note:** Support from PostgreSQL 8.2 or later.

## **Configuration parameter**

The specified configuration parameter to be set to the specified value when the function is entered, and then restored to its prior value when the function exits.


**Note:** Support from PostgreSQL 8.2 or later.

## PostgreSQL Operators

PostgreSQL supports left unary, right unary, and binary operators. Operators can be overloaded.



At least one of *LEFTARG* and *RIGHTARG* must be defined. For binary operators, both must be defined. For right unary operators, only *LEFTARG* should be defined, while for left unary operators only *RIGHTARG* should be defined.

**Note:** *LEFTARG* = Left type; *RIGHTARG* = Right type.




Just simply click  -> Operator to open an object pane for **Operator**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected operator.

### Create Operator

To create a new operator



- Select anywhere on the object pane.
- Click the  **New Operator** from the object pane toolbar.  
or
- Right-click and select  **New Operator** from the popup menu.
- Edit operator properties on the appropriate tabs of the Operator Designer.

To create a new operator with modification as one of the existing operator

- Select the operator for modifying in the object pane.
- Right-click and select the  **Design Operator** from the popup menu or simply double-click the operator.  
or
- Click the  **Design Operator** from the object pane toolbar.
- Modify operator properties on the appropriate tabs of the Operator Designer.
- Click  **Save As**.

## Edit Operator

To edit the existing operator (manage its general etc)



- Select the operator for editing in the object pane.
- Right-click and select the  **Design Operator** from the popup menu or simply double-click the operator.  
or
- Click the  **Design Operator** from the object pane toolbar.
- Edit operator properties on the appropriate tabs of the Operator Designer.

To change the name of the operator

- Select the operator for editing in the object pane.
- Right-click and select the **Rename** from the popup menu.

## Delete Operator

To delete an operator

- Select the operator for deleting in the object pane.
- Right-click and select the  **Delete Operator** from the popup menu.  
or
- Click the  **Delete Operator** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Operator Information

To achieve an operator information

- Select the operator in the object pane.
- Right-click the selected operator and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Operator Designer

**Operator Designer** is the basic Navicat tool for working with operator. It allows you to create new operator and edit the existing operator properties.

- [Editing Operator General](#)
- [Editing Advanced Operator Properties](#)
- Editing Operator Comment
- Operator SQL Preview

## Editing PostgreSQL Operator General

### Owner

The owner of the operator function.

**Note:** Support from PostgreSQL 8.0 or later.

### Schema of left type and Left type

The data type of the operator's left operand, if any. This option would be omitted for a left-unary operator.

### Schema of right type and Right type

The data type of the operator's right operand, if any. This option would be omitted for a right-unary operator.

### Schema of operator function and Operator function

The function used to implement this operator.

## Editing Advanced PostgreSQL Operator Properties

### Schema of restrict function and Restrict function

The restriction selectivity estimator function for this operator.

### Schema of join function and Join function

The join selectivity estimator function for this operator.

### Schema of commutator and Commutator

The commutator of this operator.

### Schema of negator and Negator

The negator of this operator.

#### Hash

The operator can support a hash join if this option on.

#### Merge

The operator can support a merge join if this option on.

## Additional information for PostgreSQL version below 8.3

### Schema of left sort operator and Left sort operator

If this operator can support a merge join, the left sort operator that sorts the left-hand data type of this operator.

### Schema of right sort operator and Right sort operator

If this operator can support a merge join, the right sort operator that sorts the right-hand data type of this operator.

### Schema of less than operator and Less than operator

If this operator can support a merge join, the less-than operator that compares the input data types of this operator.


### Schema of greater than operator and Greater than operator

If this operator can support a merge join, the greater-than operator that compares the input data types of this operator.

## PostgreSQL Operator Classes



An operator class defines how a particular data type can be used with an index. The operator class specifies that certain operators will fill particular roles or "strategies" for this data type and this index method. The operator class also specifies the support procedures to be used by the index method when the operator class is selected for an index column. All the operators and functions used by an operator class must be defined before the operator class is created.

**Note:** Two operator classes in the same schema can have the same name only if they are for different index methods.




Just simply click -> Operator Class to open an object pane for **Operator Class**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected operator class.

### Create Operator Class

To create a new operator class



- Select anywhere on the object pane.
- Click the  **New Operator Class** from the object pane toolbar.  
or
- Right-click and select  **New Operator Class** from the popup menu.
- Edit operator class properties on the appropriate tabs of the Operator Class Designer.

To create a new operator class with modification as one of the existing operator class

- Select the operator class for modifying in the object pane.
- Right-click and select the  **Design Operator Class** from the popup menu or simply double-click the operator class.  
or
- Click the  **Design Operator Class** from the object pane toolbar.
- Modify operator class properties on the appropriate tabs of the Operator Class Designer.
- Click  **Save As**.

## Edit Operator Class

To edit the existing operator class(manage its general, operators etc)



- Select the operator class for editing in the object pane.
- Right-click and select the  **Design Operator Class** from the popup menu or simply double-click the operator class.  
or
- Click the  **Design Operator Class** from the object pane toolbar.
- Edit operator class properties on the appropriate tabs of the Operator Class Designer.

To change the name of the operator class

- Select the operator class for editing in the object pane.
- Right-click and select the **Rename** from the popup menu.

## Delete Operator Class

To delete an operator class

- Select the operator class for deleting in the object pane.
- Right-click and select the  **Delete Operator Class** from the popup menu.  
or
- Click the  **Delete Operator Class** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Operator Class Information

To achieve an operator class information

- Select the operator class in the object pane.
- Right-click the selected operator class and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Operator Class Designer

**Operator Class Designer** is the basic Navicat tool for working with operator class. It allows you to create new operator class and edit the existing operator class properties.

- [Editing Operator Class General](#)
- [Editing Operator Class Operators](#)
- [Editing Operator Class Functions](#)
- Editing Operator Class Comment (Support from PostgreSQL 8.0 or later)
- Operator Class SQL Preview

## Editing PostgreSQL Operator Class General

### Owner

The owner of the operator class function.

**Note:** Support from PostgreSQL 8.0 or later.

### Schema of data type and Data Type

The column data type that this operator class is for.

### Index method

The name of the index method this operator class is for.

### Schema of storage type and Storage type

The data type actually stored in the index. Normally this is the same as the column data type, but some index methods (*GIN* and *GiST* for now) allow it to be different. The *STORAGE* clause must be omitted unless the index method allows a different type to be used.

### Operator family

The name of the existing operator family to add this operator class to. If not specified, a family named the same as the operator class is used (creating it, if it doesn't already exist).

**Note:** Support from PostgreSQL 8.3 or later.

### Default operator class

With this option selected, the operator class will become the default operator class for its data type. At most one operator class can be the default for a specific data type and index method.

## Editing PostgreSQL Operator Class Operators

### Strategy number

The index method's strategy number for an operator associated with the operator class.

### Schema of operator and Operator name

The operator associated with the operator class.

### Recheck

With this option selected, the index is "lossy" for this operator, and so the rows retrieved using the index must be rechecked to verify that they actually satisfy the qualification clause involving this operator.

**Note:** Before PostgreSQL 8.4, the OPERATOR clause could include a RECHECK option. This is no longer supported because whether an index operator is "lossy" is now determined on-the-fly at runtime. This allows efficient handling of cases where an operator might or might not be lossy.

## **Editing PostgreSQL Operator Class Functions**

### **Support number**


The index method's support procedure number for a function associated with the operator class.

### **Schema of function and Function name**

The function that is an index method support procedure for the operator class.



## PostgreSQL Sequences

Sequence involves creating and initializing a new special single-row table. It is usually used to generate unique identifiers for rows of a table.




Just simply click -> Sequence to open an object pane for **Sequence**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected sequence.

### Create Sequence

To create a new sequence



- Select anywhere on the object pane.
- Click the  **New Sequence** from the object pane toolbar.  
or
- Right-click and select  **New Sequence** from the popup menu.
- Edit sequence properties on the appropriate tabs of the Sequence Designer.

To create a new sequence with modification as one of the existing sequence

- Select the sequence for modifying in the object pane.
- Right-click and select the  **Design Sequence** from the popup menu or simply double-click the sequence.  
or
- Click the  **Design Sequence** from the object pane toolbar.
- Modify sequence properties on the appropriate tabs of the Sequence Designer.
- Click  **Save As**.

### Edit Sequence

To edit the existing sequence(manage its general etc)



- Select the sequence for editing in the object pane.
- Right-click and select the  **Design Sequence** from the popup menu or simply double-click the sequence.  
or
- Click the  **Design Sequence** from the object pane toolbar.
- Edit sequence properties on the appropriate tabs of the Sequence Designer.

To change the name of the sequence

- Select the sequence for editing in the object pane.
- Right-click and select the **Rename** from the popup menu.

## Delete Sequence

To delete a sequence

- Select the sequence for deleting in the object pane.
- Right-click and select the  **Delete Sequence** from the popup menu.  
or
- Click the  **Delete Sequence** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Sequence Information

To achieve a sequence information

- Select the sequence in the object pane.
- Right-click the selected sequence and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Sequence Designer

**Sequence Designer** is the basic Navicat tool for working with sequence. It allows you to create new sequence and edit the existing sequence properties.

- [Editing Sequence General](#)
- Editing Sequence Comment
- Sequence SQL Preview

## Editing PostgreSQL Sequence General

### Owner

The owner of the sequence function.

**Note:** Support from PostgreSQL 8.0 or later.

### Increment

Specifies which value is added to the current sequence value to create a new value. A positive value will make an ascending sequence, a negative one a descending sequence. The default value is 1.

### Current value

The starting value of the sequence.

### Minimum

Determines the minimum value a sequence can generate. If no minimum value is specified, then defaults will be used.

### Maximum

Determines the maximum value for the sequence. If no maximum value is specified, then default values will be used.

### Cache

Specifies how many sequence numbers are to be preallocated and stored in memory for faster access. The minimum value is 1 (only one value can be generated at a time, i.e., no cache), and this is also the default.

### Cycled

This option allows the sequence to wrap around when the maxvalue or minvalue has been reached by an ascending or descending sequence respectively. If the limit is reached, the next number generated will be the minvalue maxvalue, respectively. Otherwise, any calls to nextval after the sequence has reached its maximum value will return an error.


## **Add owned by**

Choose the **Owned by table** and **Owned by column** so that the sequence is associated with a specific table column, such that if that column (or its whole table) is dropped, the sequence will be automatically dropped as well. The specified table must have the same owner and be in the same schema as the sequence.

**Note:** Support from PostgreSQL 8.2 or later.

## PostgreSQL Types




Type registers a new data type for use in the current database. If a schema name is given then the type is created in the specified schema. Otherwise it is created in the current schema. The type name must be distinct from the name of any existing type or domain in the same schema. (Because tables have associated data types, the type name must also be distinct from the name of any existing table in the same schema.)

Just simply click -> Type to open an object pane for **Type**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected type.




**Note:** Enum Type was added in PostgreSQL 8.3.

### Create Type

To create a new type



- Select anywhere on the object pane.
- Click the  **New Type** from the object pane toolbar together with the  down arrow to choose **New Base Type / New Composite Type / New Enum Type**.  
or
- Right-click and select  **New Type** -> **New Base Type / New Composite Type / New Enum Type** from the popup menu.
- Edit type properties on the appropriate tabs of the Type Designer.

To create a new type with modification as one of the existing type

- Select the type for modifying in the object pane.
- Right-click and select the  **Design Type** from the popup menu or simply double-click the type.  
or
- Click the  **Design Type** from the object pane toolbar.
- Modify type properties on the appropriate tabs of the Type Designer.
- Click  **Save As**.



## Edit Type

To edit the existing type(manage its general etc)

- Select the type for editing in the object pane.
- Right-click and select the  **Design Type** from the popup menu or simply double-click the type.  
or
- Click the  **Design Type** from the object pane toolbar.
- Edit type properties on the appropriate tabs of the Type Designer.

## Delete Type

To delete a type

- Select the type for deleting in the object pane.
- Right-click and select the  **Delete Type** from the popup menu.  
or
- Click the  **Delete Type** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Type Information

To achieve a type information

- Select the type in the object pane.
- Right-click the selected type and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Type Designer

**Type Designer** is the basic Navicat tool for working with type. It allows you to create new type and edit the existing type properties.

- [Editing Base Type Properties](#)
- [Editing Composite Type Properties](#)
- [Editing Enum Type Properties](#)
- Editing Type Comment
- Type SQL Preview

## Editing PostgreSQL Base Type Properties

**Base types** are those, like `int4`, that are implemented below the level of the SQL language (typically in a low-level language such as C). They generally correspond to what are often known as abstract data types. PostgreSQL can only operate on such types through functions provided by the user and only understands the behavior of such types to the extent that the user describes them. Base types are further subdivided into scalar and array types. For each scalar type, a corresponding array type is automatically created that can hold variable-size arrays of that scalar type.

- [Editing Base Type General](#)
- [Editing Advanced Base Type Properties](#)

## Editing PostgreSQL Base Type General

### Input Schema and Input

The function that converts data from the type's external textual form to its internal form.

### Output Schema and Output

The function that converts data from the type's internal form to its external textual form.

### Length

A numeric constant that specifies the length in bytes of the new type's internal representation. The default assumption is that it is variable-length.

#### Variable

Checks this option if the type length is unknown.

### Default

The default value for the data type. If this is omitted, the default is null.

### Element

The type being created is an array; this specifies the type of the array elements.

### Delimiter

The delimiter character to be used between values in arrays made of this type.

### Alignment

The storage alignment requirement of the data type. If specified, it must be char, int2, int4, or double; the default is int4.

### Storage

The storage strategy for the data type. If specified, must be plain, external, extended, or main; the default is plain.

#### Pass by value

Indicates that values of this data type are passed by value rather than by reference.

### Owner

The owner of the type.

**Note:** Support from PostgreSQL 8.0 or later.

## **Editing Advanced PostgreSQL Base Type Properties**

The **Advanced** tab is supported from PostgreSQL 7.4 or later.

### **Receive Schema and Receive**

The function that converts data from the type's external binary form to its internal form.

### **Send Schema and Send**

The function that converts data from the type's internal form to its external binary form.

### **Analyze Schema and Analyze**

The function that performs statistical analysis for the data type.

**Note:** Support from PostgreSQL 8.0 or later.

### **Type Modifier Input Schema and Type Modifier Input**

The function that converts an array of modifier(s) for the type into internal form.

**Note:** Support from PostgreSQL 8.3 or later.

### **Type Modifier Output Schema and Type Modifier Output**

The function that converts the internal form of the type's modifier(s) to external textual form.

**Note:** Support from PostgreSQL 8.3 or later.

## Editing PostgreSQL Composite Type Properties

**Composite types**, or row types, are created whenever the user creates a table; it's also possible to define a "stand-alone" composite type with no associated table. A composite type is simply a list of base types with associated field names. A value of a composite type is a row or record of field values. The user can access the component fields from SQL queries.

- [Editing Composite Type General](#)

## Editing PostgreSQL Composite Type General

### Name

The name of an attribute (column) for the composite type.

### Type

The name of an existing data type to become a column of the composite type.

### Length and Scale

Use the **Length** edit box to define the length of the field and use **Scale** edit box to define the number of digits after the decimal point. (if required for the selected data type)

### Dimensions

The dimensions of array specifiers.

### Owner

The owner of the type.

**Note:** Support from PostgreSQL 8.0 or later.

## Editing PostgreSQL Enum Type Properties

**Enumerated (Enum) types** are data types that are comprised of a static, predefined set of values with a specific order. They are equivalent to the enum types in a number of programming languages. An example of an enum type might be the days of the week, or a set of status values for a piece of data.

**Note:** Enum Type was added in PostgreSQL 8.3.

- [Editing Enum Type General](#)

## **Editing PostgreSQL Enum Type General**

### **Label**


A string literal representing the textual label associated with one value of an enum type.

### **Owner**

The owner of the type.

## PostgreSQL Tablespaces



A tablespace allows superusers to define an alternative location on the file system where the data files containing database objects (such as tables and indexes) may reside.

Just simply click -> Tablespace to open an object pane for **Tablespace**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected tablespace.

**Note:** Tablespace was added in PostgreSQL 8.0.



### Create Tablespace

To create a new tablespace

- Select anywhere on the object pane.
- Click the  **New Tablespace** from the object pane toolbar.  
or
- Right-click and select  **New Tablespace** from the popup menu.
- Edit tablespace properties on the appropriate tabs of the Tablespace Designer.

### Edit Tablespace

To edit the existing tablespace (manage its general, privileges etc)



- Select the tablespace for editing in the object pane.
- Right-click and select the  **Design Tablespace** from the popup menu or simply double-click the tablespace.  
or
- Click the  **Design Tablespace** from the object pane toolbar.
- Edit tablespace properties on the appropriate tabs of the Tablespace Designer.

To change the name of the tablespace

- Select the tablespace for editing in the object pane.
- Right-click and select the **Rename** from the popup menu.

## Delete Tablespace

To delete a tablespace

- Select the tablespace for deleting in the object pane.
- Right-click and select the  **Delete Tablespace** from the popup menu.  
or
- Click the  **Delete Tablespace** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Tablespace Information

To achieve a tablespace information

- Select the tablespace in the object pane.
- Right-click the selected tablespace and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Tablespace Designer

**Tablespace Designer** is the basic Navicat tool for working with tablespace. It allows you to create new tablespace and edit the existing tablespace properties.

- [Editing Tablespace General](#)
- Editing Tablespace Comment (Support from PostgreSQL 8.2 or later)
- Tablespace SQL Preview

## Editing PostgreSQL Tablespace General

### Location


The directory that will be used for the tablespace. The directory must be empty and must be owned by the PostgreSQL system user. The directory must be specified by an absolute path name.

### Owner

The name of the user who will own the tablespace. If omitted, defaults to the user executing the command. Only superusers may create tablespaces, but they can assign ownership of tablespaces to non-superusers.



## PostgreSQL Casts

A cast specifies how to perform a conversion between two data types.




Just simply click -> Cast to open an object pane for **Cast**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected cast.

### Create Cast

To create a new cast


- Select anywhere on the object pane.
- Click the  **New Cast** from the object pane toolbar.  
or
- Right-click and select  **New Cast** from the popup menu.
- Edit cast properties on the appropriate tabs of the Cast Designer.

To create a new cast with modification as one of the existing cast

- Select the cast for modifying in the object pane.
- Right-click and select the  **Design Cast** from the popup menu or simply double-click the cast.  
or
- Click the  **Design Cast** from the object pane toolbar.
- Modify cast properties on the appropriate tabs of the Cast Designer.
- Click  **Save As**.



### Edit Cast

To edit the existing cast(manage its general etc)

- Select the cast for editing in the object pane.
- Right-click and select the  **Design Cast** from the popup menu or simply double-click the cast.  
or
- Click the  **Design Cast** from the object pane toolbar.
- Edit cast properties on the appropriate tabs of the Cast Designer.

## Delete Cast

To delete a cast

- Select the cast for deleting in the object pane.
- Right-click and select the  **Delete Cast** from the popup menu.  
or
- Click the  **Delete Cast** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Cast Information

To achieve a cast information

- Select the cast in the object pane.
- Right-click the selected cast and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Cast Designer

**Cast Designer** is the basic Navicat tool for working with cast. It allows you to create new cast and edit the existing cast properties.

- [Editing Cast General](#)
- Editing Cast Comment (Support from PostgreSQL 8.0 or later)
- Cast SQL Preview

## Editing PostgreSQL Cast General

### Schema of source type and Source type

The schema and name of the source data type of the cast.

### Schema of target type and Target type

The schema and name of the target data type of the cast.

### Schema of function and Function

The function used to perform the cast. The function name may be schema-qualified. If it is not, the function will be looked up in the schema search path. The function's result data type must match the target type of the cast.

If no function is specify, indicates that the source type and the target type are binary compatible, so no function is required to perform the cast.

#### **Implicit**


Indicates that the cast may be invoked implicitly in any context.

#### **Assignment**

Indicates that the cast can be invoked implicitly in assignment contexts.



## PostgreSQL Languages

Language can register a new procedural language with a PostgreSQL database. Subsequently, functions and trigger procedures can be defined in this new language. The user must have the PostgreSQL superuser privilege to register a new language.




Just simply click -> Language to open an object pane for **Language**. A right-click displays the popup menu or using the object pane toolbar, allowing you to create new, edit and delete the selected Language.

### Create Language

To create a new language



- Select anywhere on the object pane.
- Click the  **New Language** from the object pane toolbar.  
or
- Right-click and select  **New Language** from the popup menu.
- Edit language properties on the appropriate tabs of the Language Designer.

To create a new language with modification as one of the existing language

- Select the language for modifying in the object pane.
- Right-click and select the  **Design Language** from the popup menu or simply double-click the language.  
or
- Click the  **Design Language** from the object pane toolbar.
- Modify language properties on the appropriate tabs of the Language Designer.
- Click  **Save As**.

### Edit Language

To edit the existing language(manage its properties, privileges etc)

- Select the language for editing in the object pane.
- Right-click and select the  **Design Language** from the popup menu or simply double-click the language.  
or
- Click the  **Design Language** from the object pane toolbar.
- Edit language properties on the appropriate tabs of the Language Designer.



To change the name of the language

- Select the language for editing in the object pane.
- Right-click and select the **Rename** from the popup menu.

**Note:** Support from PostgreSQL 7.4 or later.

## Delete Language

To delete a language

- Select the language for deleting in the object pane.
- Right-click and select the  **Delete Language** from the popup menu.  
or
- Click the  **Delete Language** from the object pane toolbar.
- Confirm deleting in the dialog window.

## Achieve Language Information

To achieve a language information

- Select the language in the object pane.
- Right-click the selected language and choose **Object Information** from the popup menu.  
or
- Choose View -> Object Information in the main menu.

## PostgreSQL Language Designer

**Language Designer** is the basic Navicat tool for working with language. It allows you to create new language and edit the existing language properties.

- [Editing Language General](#)
- Editing Language Comment (Support from PostgreSQL 8.0 or later)
- Language SQL Preview

## Editing PostgreSQL Language General

### Owner

The owner of the language.

**Note:** Support from PostgreSQL 8.3 or later.

### Schema of handler and Handler

Call Handler is the name of a previously registered function that will be called to execute the procedural language functions. The call handler for a procedural language must be written in a compiled language such as C with version 1 call convention and registered with PostgreSQL as a function taking no arguments and returning the *language\_handler* type, a placeholder type that is simply used to identify the function as a call handler.

### Schema of validator and Validator

Validator function is the name of a previously registered function that will be called when a new function in the language is created, to validate the new function. If no validator function is specified, then a new function will not be checked when it is created. The validator function must take one argument of type oid, which will be the OID of the to-be-created function, and will typically return void.

A validator function would typically inspect the function body for syntactical correctness, but it can also look at other properties of the function, for example if the language cannot handle certain argument types. To signal an error, the validator function should use the ereport() function. The return value of the function is ignored.

### Trusted

Specifies that the call handler for the language is safe, that is, it does not offer an unprivileged user any functionality to bypass access restrictions. If this key word is omitted when registering the language, only users with the PostgreSQL superuser privilege can use this language to create new functions.