Database Design with MySQL Workbench

A Round Trip Software Engineering Case

Jerzy Letkowski

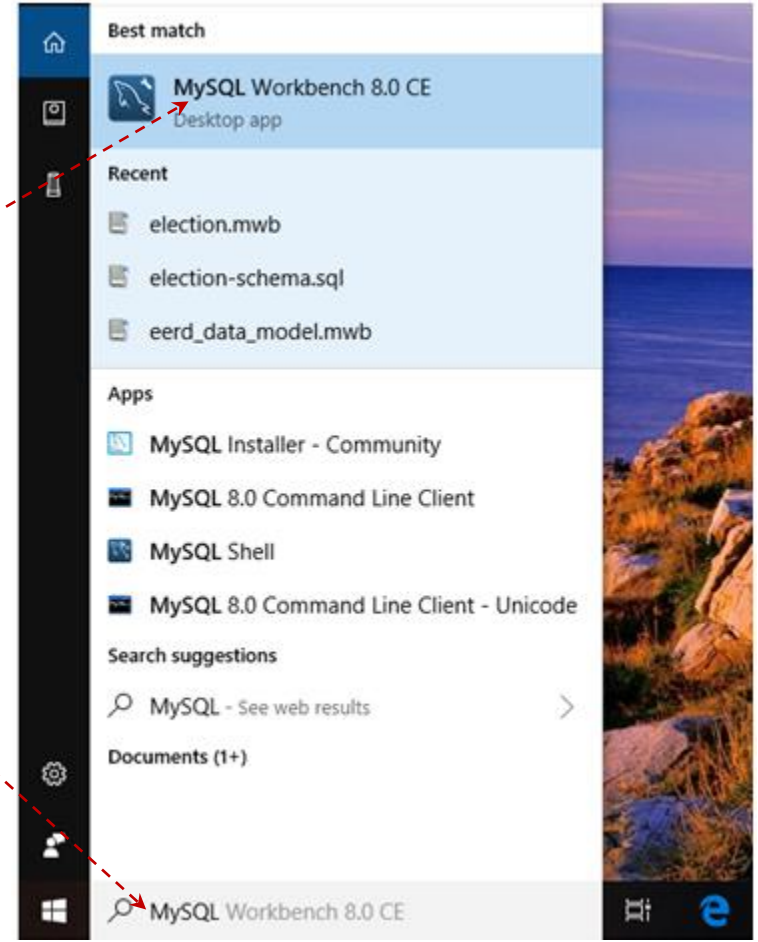
Prerequisites - Downloads

- MySQL Server and MySQL Workbench are expected to be installed. The setup program(s) can be downloaded from the MySQL Website.
- This instruction uses the server and workbench installed using the July 5, 2018 installer, downloaded from http://www.mysql.com/:
  - ➢ Tab: Downloads (GA)
  - ➢ [Scroll Down] MySQL Community Edition (GPL):
    Community (GPL) Downloads »
  - ➢ MySQL Community Server (GPL)
  - ➢ Windows (x86, 32, 64-bit), MySQL Installer MSI : Go to Download Page
  - ➢ [Scroll Down] Windows (x86, 32-bit), MSI Installer 8.0.11 230.0M Download
  - ➢ No thanks, just start my download.
- The **mysql-installer-community-8.0.11.0.msi** version was available, at the time of writing this instruction.
- Other major operating systems are also supported, including Mac OS X and Linux.

Prerequisites - Installation

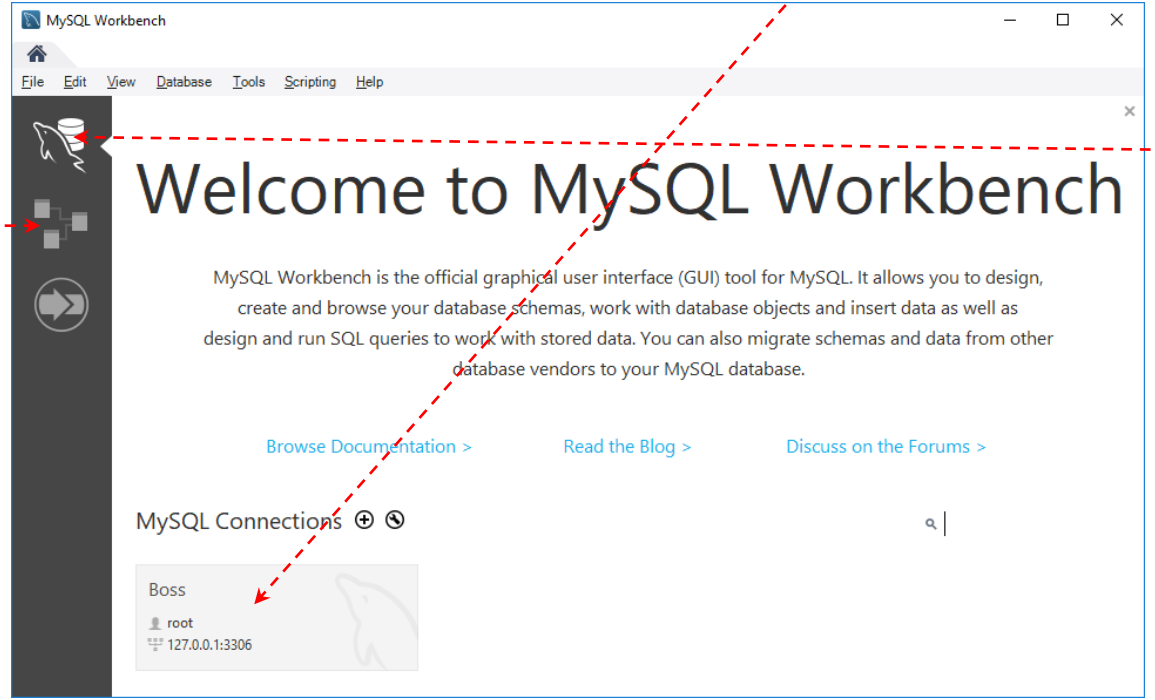
- Using the standard Window installer is pretty straight forward. Just follow the default developer's options.
- It is very important that you do not forget the **root** password.
- When done, follow the remaining slides to learn more about database design and implementation, using MySQL Workbench and Community Server.

You can ***start*** **MySQL Workbench** by using Windows search for **MySQL** and picking it from the list.

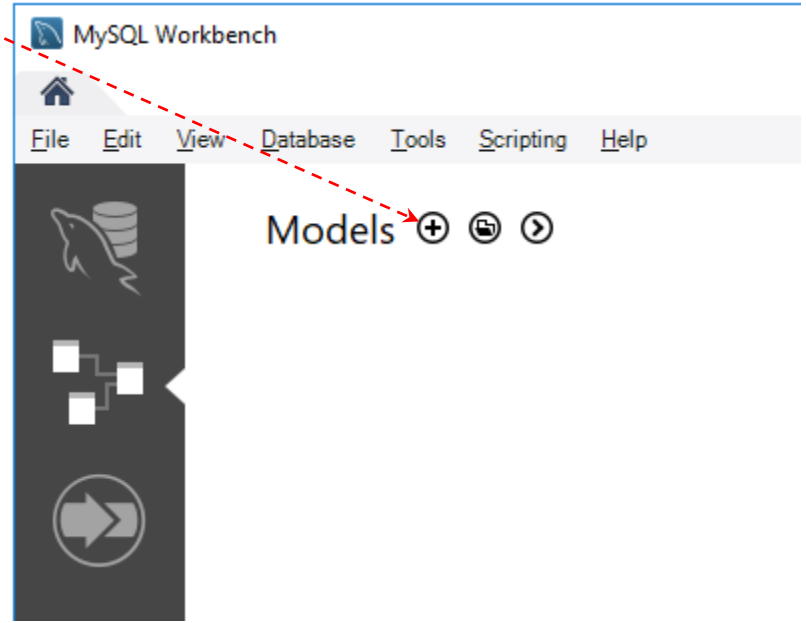


The front page of the **MySQL Workbench** program *shows* the initial (root) **connection** , the **Models**' interface.

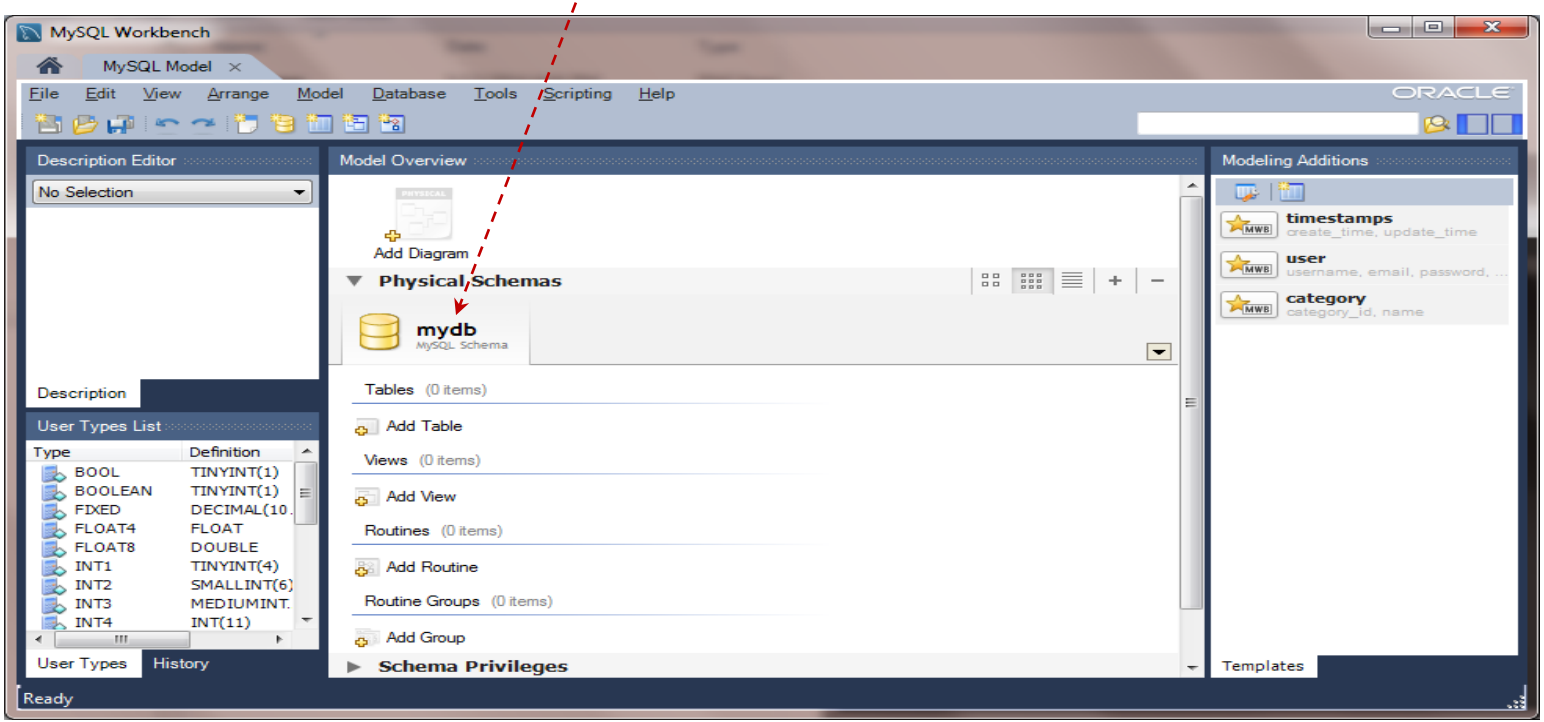
Click the **Models** icon to open the data modeling facility.



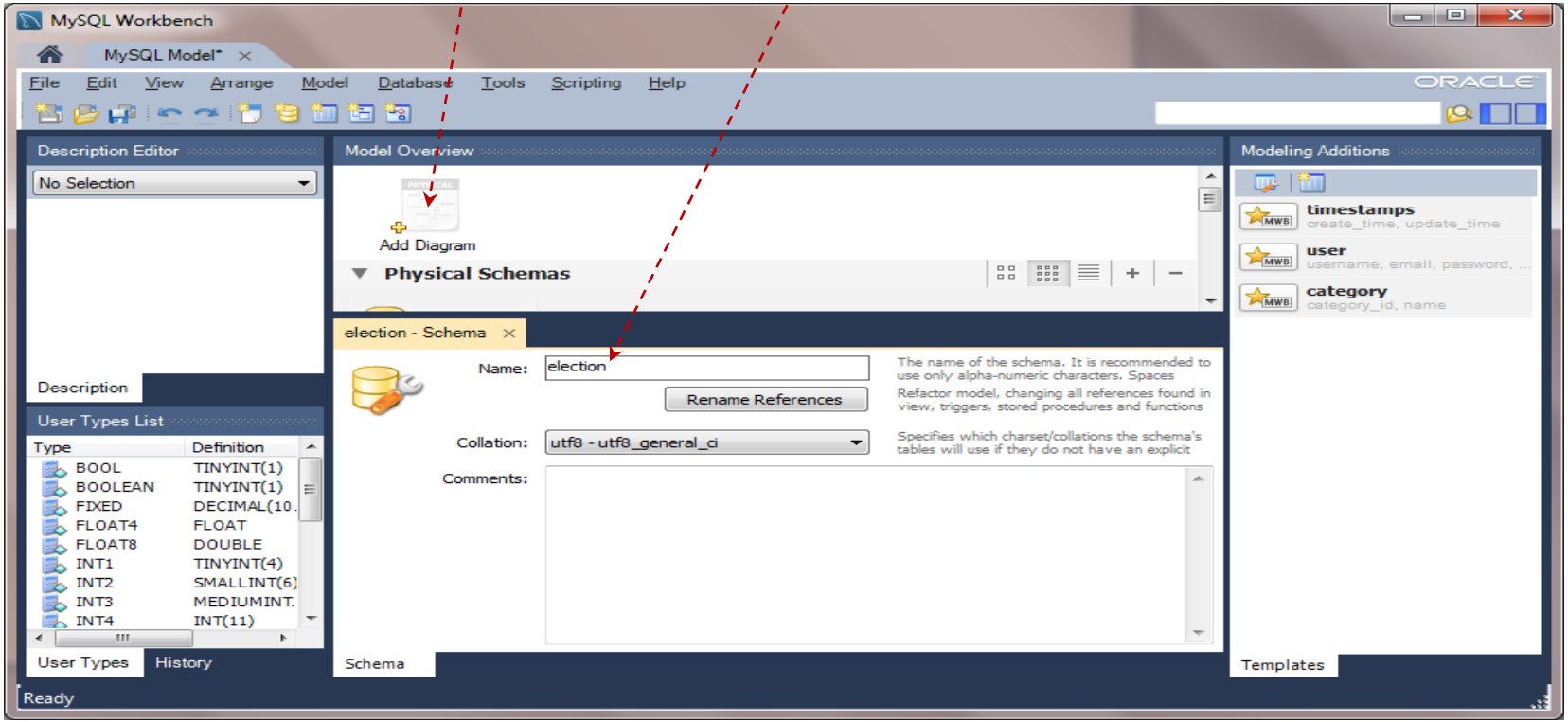
Click the **Models** (+) icon in order to create a new relational database model.



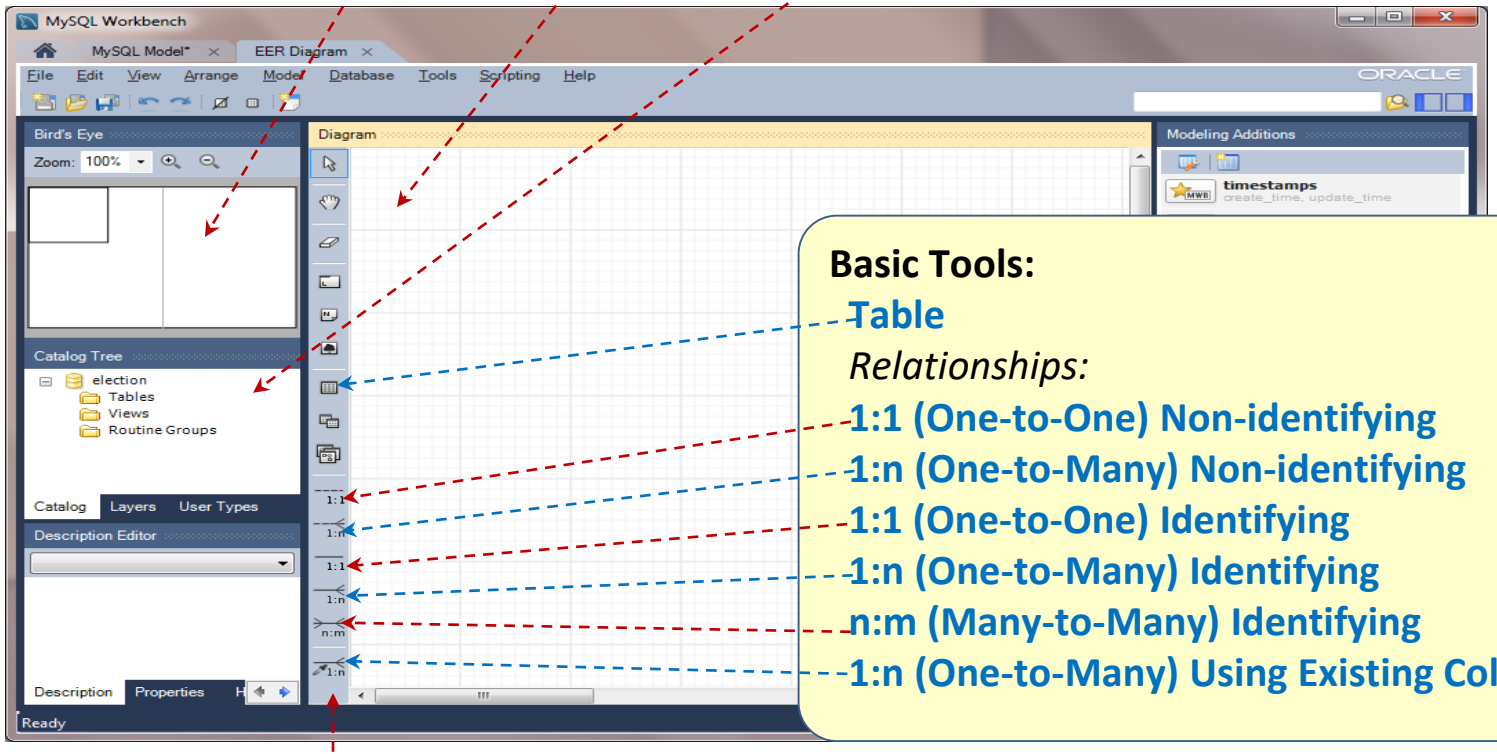
Double-click the **default schema (database) name** tab (**mydb**) in order to create a new database name.



Type name **election**, and *double-click* the **Add Diagram** icon.



MySQL Workbench ***opens*** the **Diagram canvas** along with other useful panels, including the **Birds Eye view panel**, and **Catalog Tree panel**.



Basic Tools:

- **Table**

Relationships:

- **1:1 (One-to-One) Non-identifying**
- **1:n (One-to-Many) Non-identifying**
- **1:1 (One-to-One) Identifying**
- **1:n (One-to-Many) Identifying**
- **n:m (Many-to-Many) Identifying**
- **1:n (One-to-Many) Using Existing Columns**

The **Toolbar** contains graphical tools that are used to paint **Enhanced Entity Relationship Diagrams** (**EERDs**).

Problem Statement

A Database for an Online Voting System

Our mission is to design a database for an on-line election system that will be utilized to conduct election of new leaders of a non-profit organization. The organization has members some of which hold leadership positions (president, vice president, treasurer, newsletter editor, annual meeting coordinator, secretary, etc.). Some of the members have been nominated to run for the positions. Assume that they have already accepted their nominations, thus becoming official candidates for the available positions. The organization's statute states that each candidate may only run for one position and each member may cast no more than one vote for each of the positions. The database should facilitate the voting process and record all votes assigned to the candidates but it should not tell which member has voted for which candidate.

Entities

From the problem definition one could identify the following base entities:

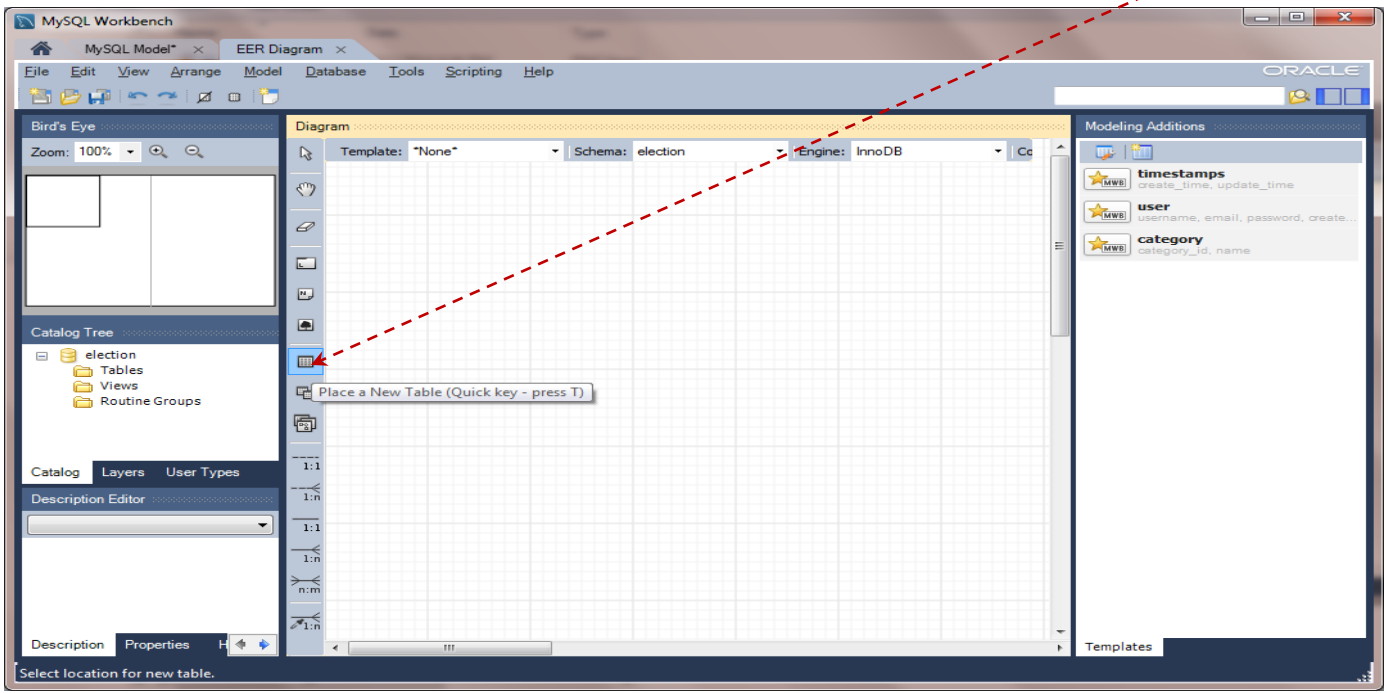
- **Member**: a list (set) of members of the organization, some of whom are candidates;
- **Office**: a list of positions ('president', 'treasurer', 'editor', etc.);

Other entities result from associations between the base entities.

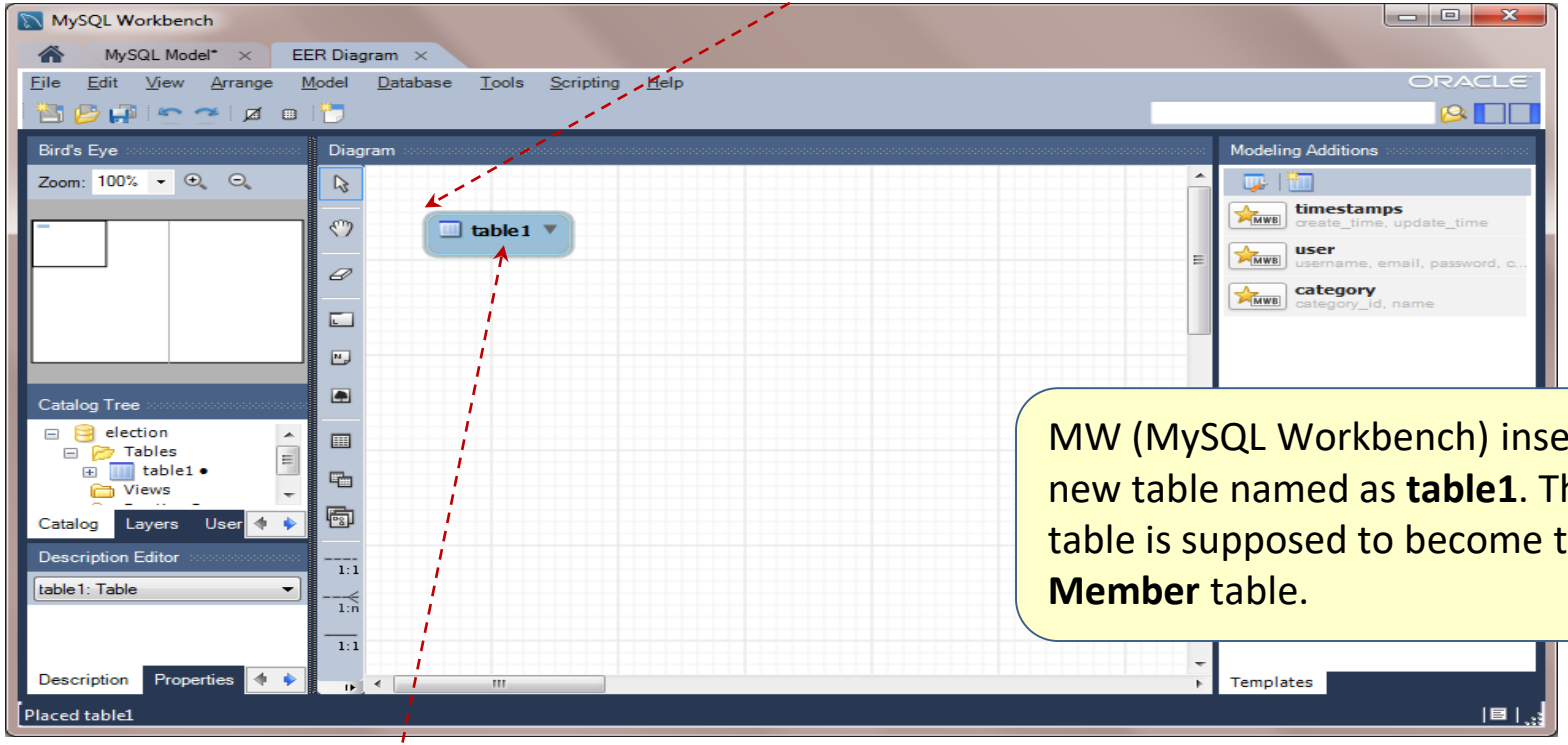
Our mission is to design a database for an on-line election system that will be utilized to conduct election of new leaders of a non-profit organization. The organization has **members** some of which hold **leadership positions** (president, vice president, treasurer, newsletter editor, annual meeting coordinator, secretary, etc.). Some of the **members have been nominated** to run for the positions. Assume that they have already accepted their nominations, thus becoming official **candidates** for the available **positions**. The organization's statute states that each **candidate** may only run for one **position** (**office**) and each **member** may cast no more than one **vote** for each of the **positions**. The database should facilitate the voting process and record all **votes** assigned to the candidates but it should not tell which member has voted for which candidate.

Entity **Member**

To add entity (table) **Member** to the data model (EER diagram) ***click*** the **New Table** icon.



Next *click* anywhere on the **Diagram canvas**.



MW (MySQL Workbench) inserts the new table named as **table1**. This table is supposed to become the **Member** table.

In order to change the name of the table and add its attributes, ***double-click*** table **table1**.

Type name **Member**.

Add column name **mid** as (Datatype) **INT**.

Set column **mid** as the **Primary Key** (PK).

Add other columns:
firstName as **VARCHAR(45)**,
lastName as **VARCHAR(90)**,
pass as **CHAR(12)**,
email as **VARCHAR(90)**.

When done, *close* the **Member - Table** definition panel.

Notice that the primary key is automatically set as **Not NULL** (**NN**). Data types **INT**, **VARCHAR** and **CHAR** stand for an integer, a variable-length string (text) and a fixed-length string, respectively.

Some individual have hard time of visualizing abstract entities. The attached Excel macro-workbook, **election.xlsm**, contains different depictions of the entities.

Each worksheet shows three representations for each of the election-database entities: **ERD** (**UML**), **SQL** and **Excel Table**.

The **election.xlsm** workbook also contains a macro-function,

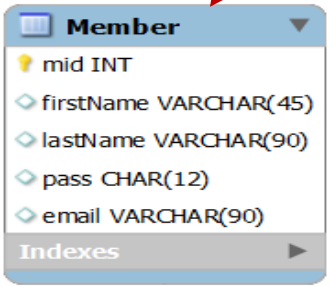
```
sqlInsert(table_name_abs_ref,data_type_abs_ref,record_ref),
```

that transforms rows of the tables into their **SQL-Insert** statements. In order to facilitate this function, a row with "high-level" data types is added (right above the column names), where **n** stands for numeric and **c**—nonnumeric.

Note that the last part of this instruction shows how to transform the **ERD** view into the **SQL** statements. Typically, when designing a new database, the **SQL** representations are not available until the **ERD** model is completed and transformed to it. At this point, **SQL** (**S**tructured **Q**uery **L**anguage) may be somewhat mysterious (it will be explored in detail later). Suffice to say, it is the official language for managing relational database systems. It is shown here for completeness.

The **Member** worksheet of the **election.xlsm** workbook shows three views of the Member entity: **UML**, **SQL Schema**, **SQL Data**, and **Excel Table**.

The `sqlInsert()` function transforms the rows of the **Excel Table** into **SQL-Insert** statements.



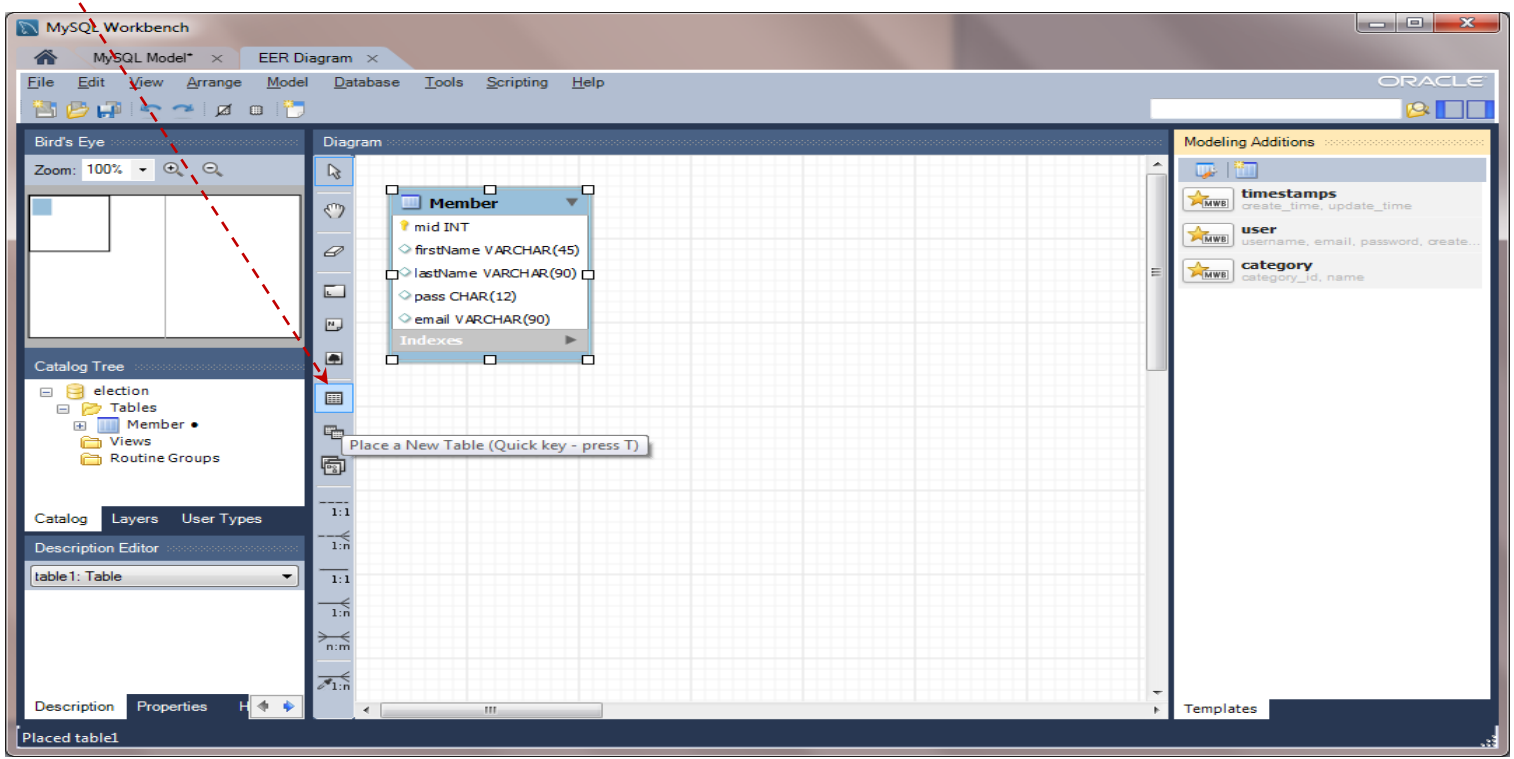

```
CREATE TABLE Member
(
    mid int primary key,
    firstName varchar(45) NOT NULL,
    lastName varchar(90) NOT NULL,
    pass char(12) NOT NULL,
    email varchar(90)
);
```

n	c	c	c	c
mid	**firstName**	**lastName**	**pass**	**email**
1	Ann	Anson	aa01119	aanson@misor.org
2	Ben	Jamin	bj11122	bjamin@misor.org
3	Cir	Cus	cc22211	ccus@misor.org
4	Don	Donski	dd00413	ddonski@misor.org
5	Eve	Lady	el98765	elady@misor.org
6	Fin	End	fe00011	fend@misor.org
7	Gin	Rum	gr12345	grum@misor.org

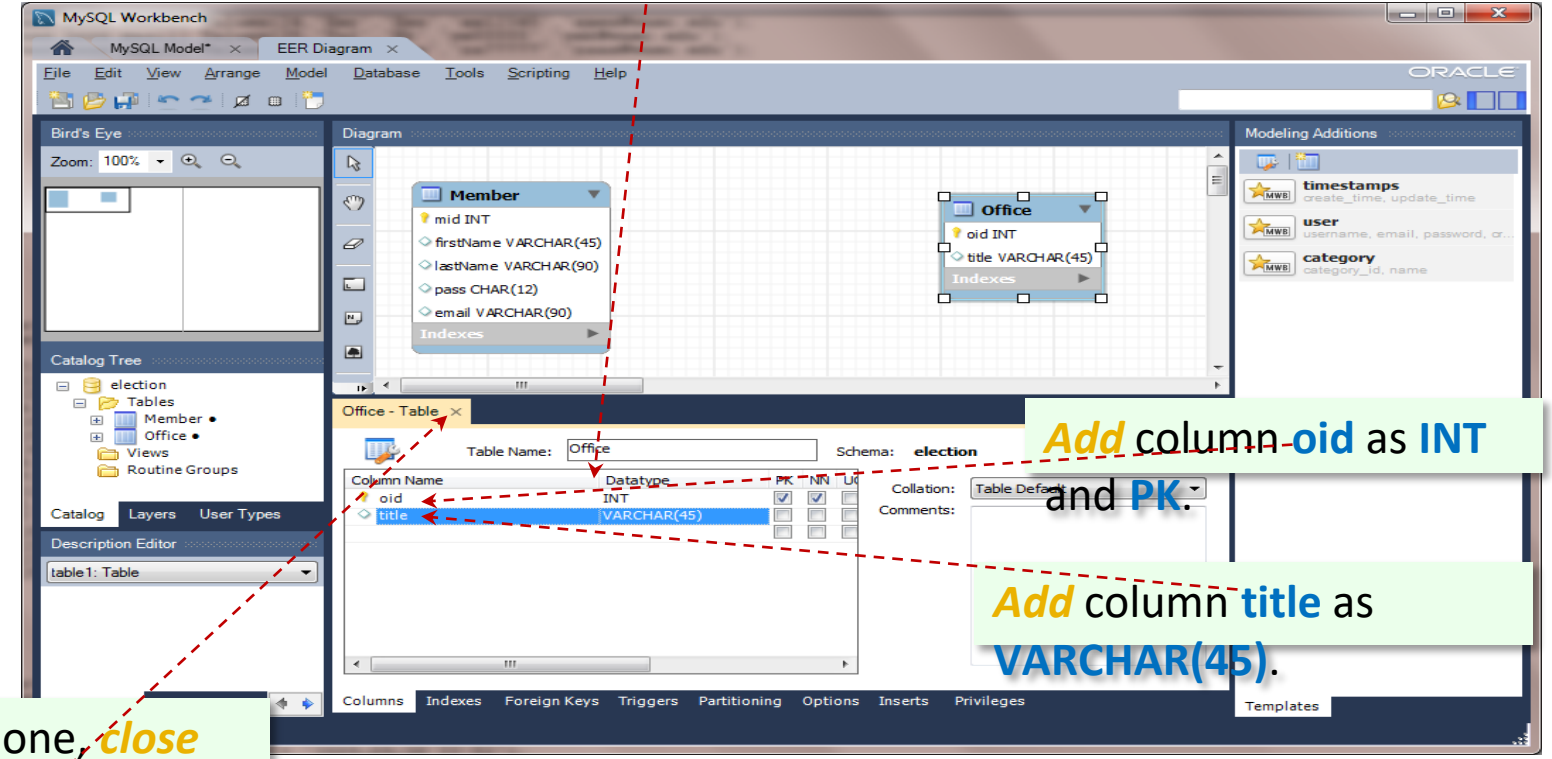
```
Insert into Member(mid,firstName,lastName,pass,email) Values(1,'Ann','Anson','aa01119','aanson@misor.org');
Insert into Member(mid,firstName,lastName,pass,email) Values(2,'Ben','Jamin','bj11122','bjamin@misor.org');
Insert into Member(mid,firstName,lastName,pass,email) Values(3,'Cir','Cus','cc22211','ccus@misor.org');
Insert into Member(mid,firstName,lastName,pass,email) Values(4,'Don','Donski','dd00413','ddonski@misor.org');
Insert into Member(mid,firstName,lastName,pass,email) Values(5,'Eve','Lady','el98765','elady@misor.org');
```

Entity **Office**

To add entity (table) Office to the data model (EER diagram) ***click*** the **New Table** icon.



Change the name of this table to **Office**.



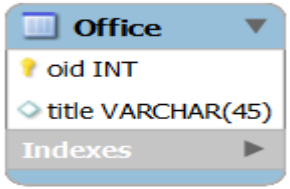
Add column **oid** as **INT** and **PK**.

Add column **title** as **VARCHAR(45)**.

When done, *close* the **Office – Table** definition panel.

The **Office** worksheet of the **election.xlsm** workbook shows three views of each of the **Office** entity: **UML**, **SQL Schema**, **SQL Data**, and **Excel Table**.

The `sqlInsert()` function transforms the rows of the **Excel Table** into **SQL-Insert** statements.




```
CREATE TABLE Office
(
    oid int primary key,
    title varchar(50) NOT NULL
);
```

Table:	Office
n	c
oid	title
1	President
2	Vice-President
3	Treasurer
4	Editor
5	Web Manager

```
Insert into Office(oid,title) Values(1,'President');
Insert into Office(oid,title) Values(2,'Vice-President');
Insert into Office(oid,title) Values(3,'Treasurer');
Insert into Office(oid,title) Values(4,'Editor');
Insert into Office(oid,title) Values(5,'Web Manager');
```

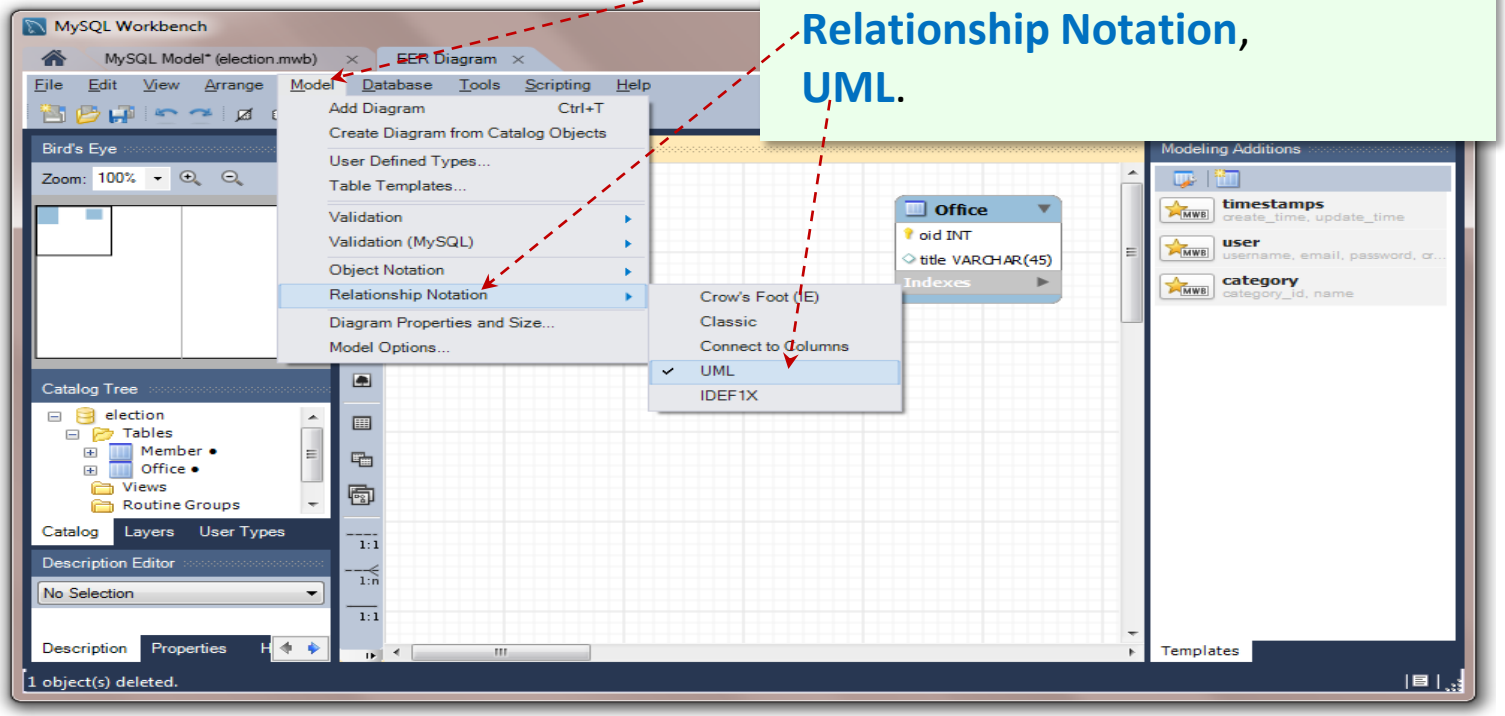
This instruction shows how to develop an **EER** diagram (**ERD**) in which relationships are modeled using the **UML** notation. Make sure that your notation is the same.

Select the following menu options:

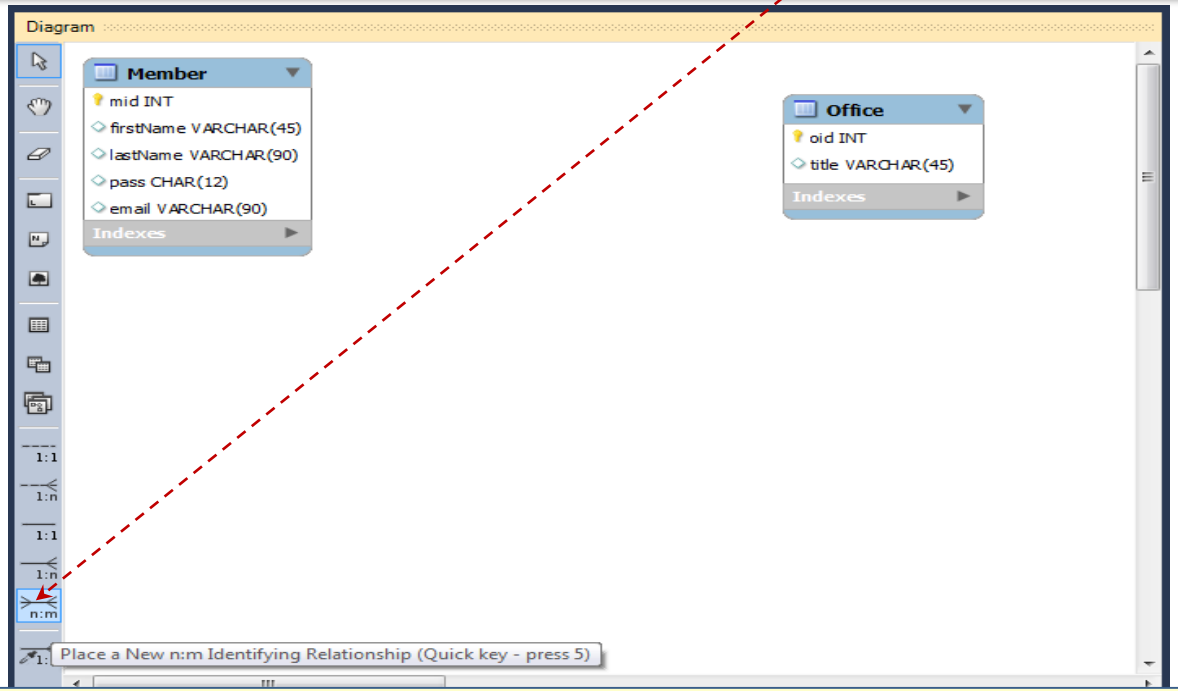
Model

Relationship Notation,

UML.



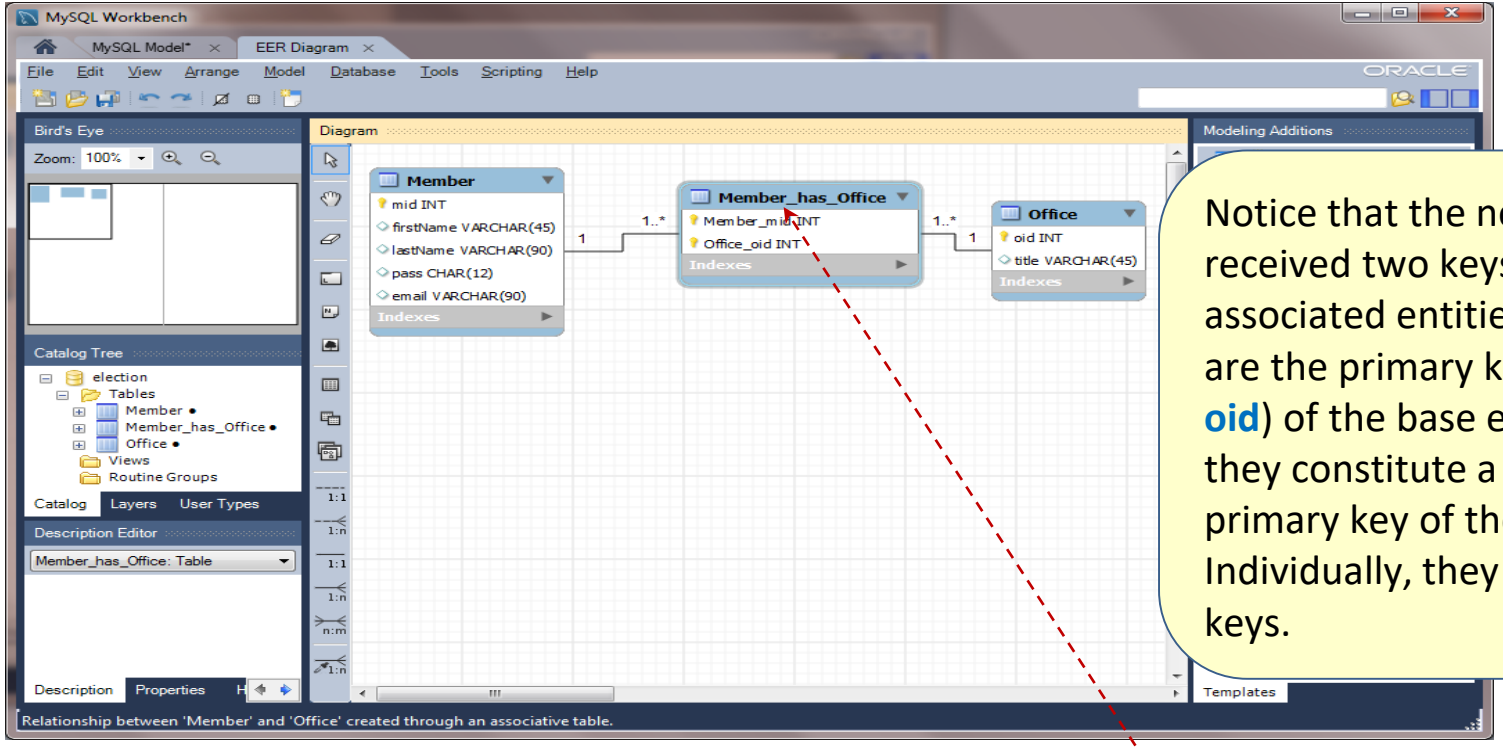
To add a relationship between entities **Member** and **Office**, ***click*** the **m:n Identifying Relationship** button.



Notice that **m:n** stands for **Many-to-Many**. Since a **member** (instance of entity **Member**) participates in election of many **offices** (positions) and a given **office** (instance of entity **Office**) is being elected by many members, this is a **Many-to-Many** relationship.

Note: Despite selecting the **UML** notation, the relationship tools are shown, using the **Crow-Foot** notation.

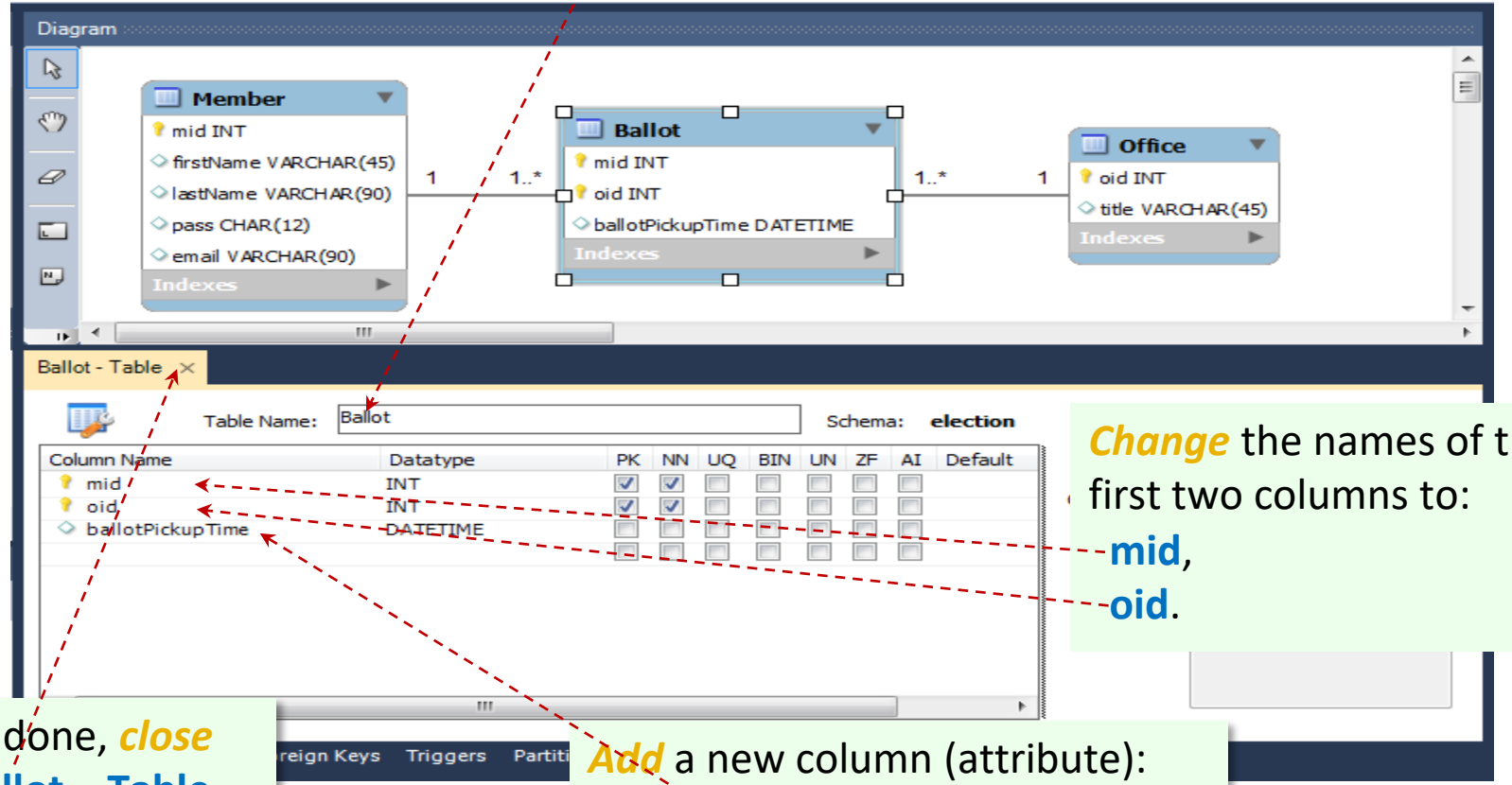
MW inserts a new [default] entity, **Member-has-Office** and sets the relationships of this entity with **Member** and **Office** to **Many-to-One**. This is a typical resolution of the **Many-to-Many** relationships as required by the relational database model.



Notice that the new entity has received two keys from the associated entities. Their sources are the primary keys (**mid** and **oid**) of the base entities. Jointly, they constitute a [composite] primary key of the new entity's . Individually, they are foreign keys.

In order to customize the **Member-has-Office** entity, ***double-click*** this **entity**.

Change the name of this entity to **Ballot**.



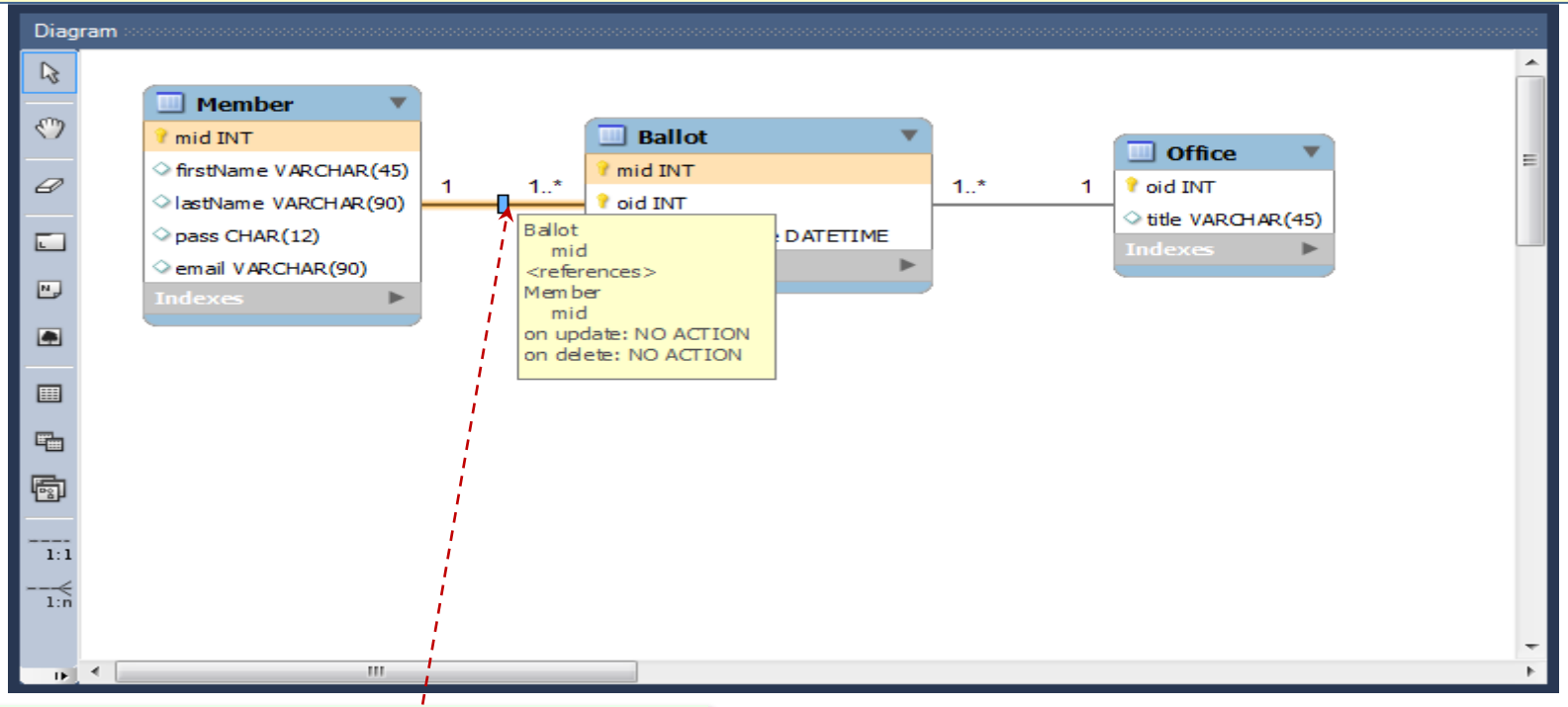
Change the names of the first two columns to:

- **mid**,
- **oid**.

Add a new column (attribute): **ballotPickupTime** as **DATETIME**.

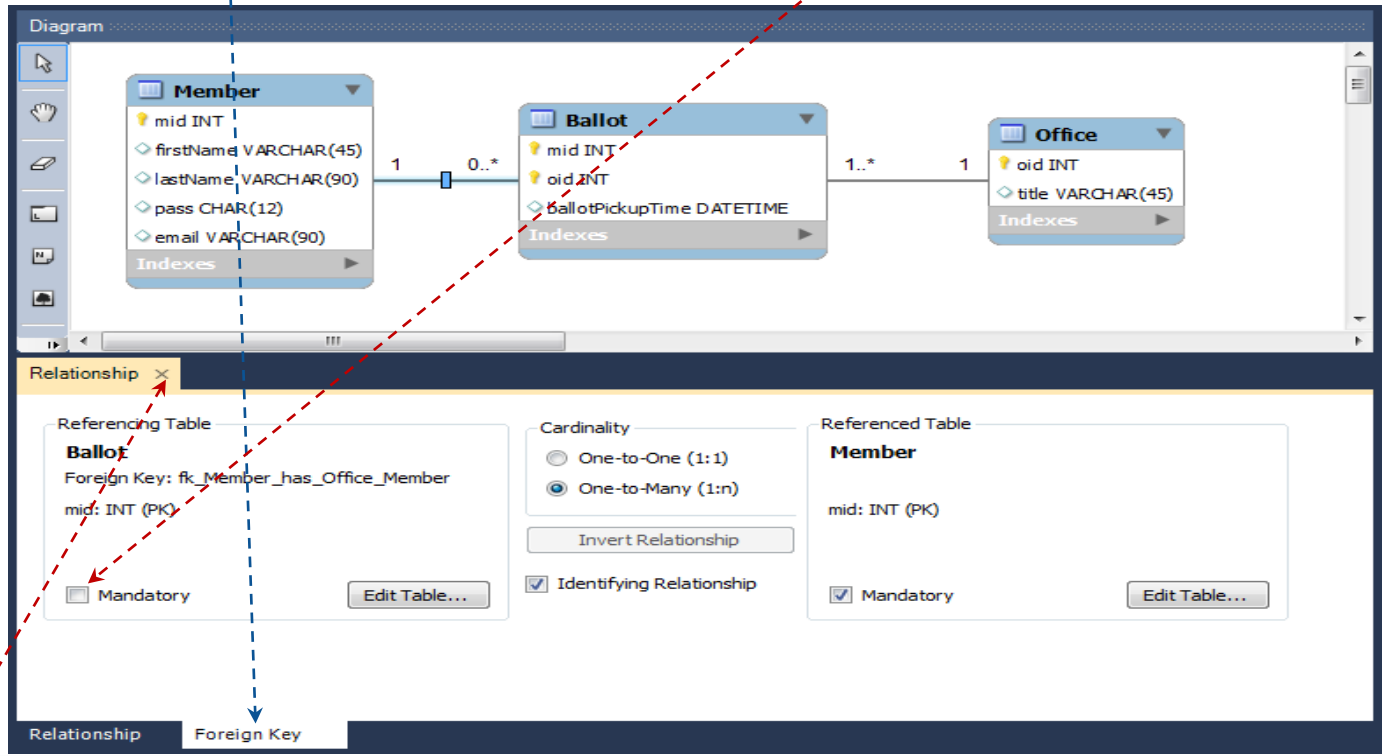
When done, ***close*** the **Ballot – Table** definition panel.

Since theoretically there may be members who will not participate in the election of their representatives (officers), the participation of the **Ballot** entity in the relationship with entity **Member** is optional. The cardinality constraint at the **Ballot** side should be changed from many-mandatory (**1..***) to many-optional (**0..***).



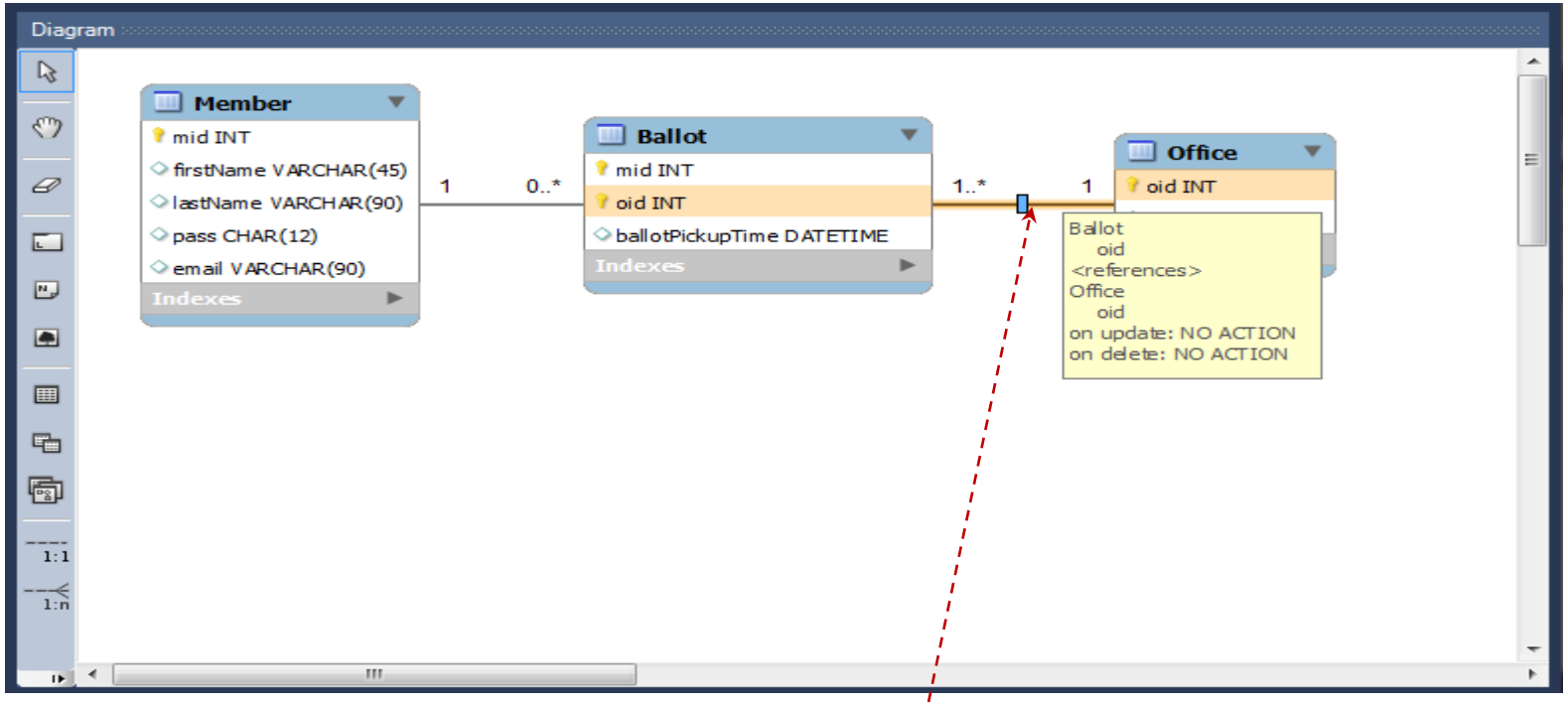
Double-click the **Member - Ballot** link.

Select the **Foreign Key** tab and *uncheck* option **Mandatory** in the **Ballot** panel.



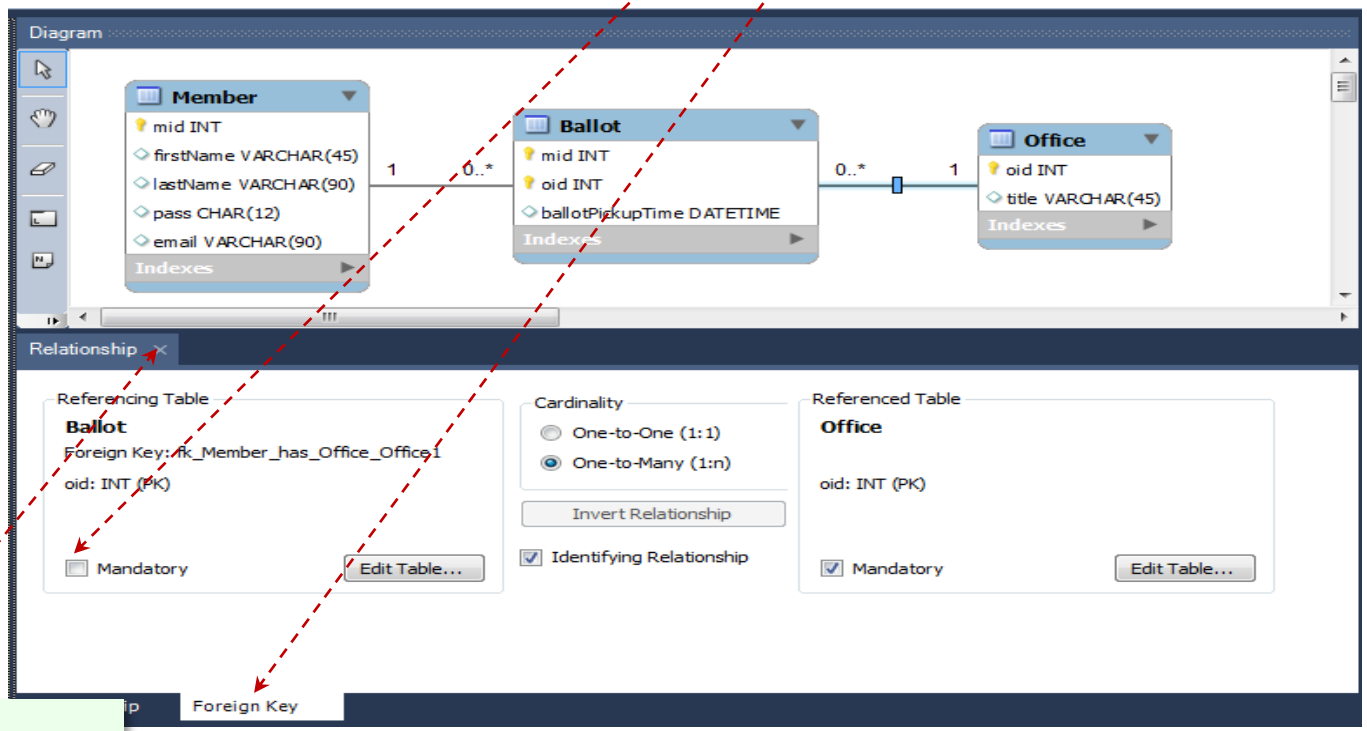
When done, *close* the **Relationship** panel.

Since theoretically there may be Office instances (positions) that will not receive any votes (at any given time), the participation of the **Ballot** entity in the relationship with entity **Office** should be optional. The cardinality constraint at the **Ballot** side should be changed from many-mandatory (**1..***) to many-optional (**0..***).



Double-click the **Ballot – Office** link.

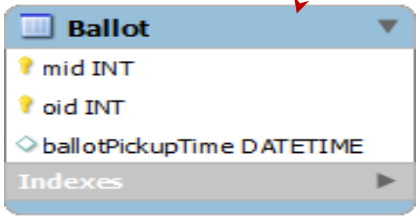
Select the **Foreign Key** tab and *uncheck* option **Mandatory** in the **Ballot** panel.



When done, *close* the **Relationship** panel.

The **Ballot** worksheet of the **election.xlsm** workbook shows three views of each of the **Ballot** entity: **UML**, **SQL** and **Excel Table**.

The `sqlInsert()` function transforms the rows of the table into **SQL-Insert** statements.




```
CREATE TABLE Ballot
(
    mid int primary key,
    oid int NOT NULL,
    ballotPickupTime DateTime,
    FOREIGN KEY (mid) REFERENCES Member(mid),
    FOREIGN KEY (oid) REFERENCES Office(oid)
);
```

```
Insert into Ballot(mid,oid,ballotPickupTime) Values(1,1,'2009-10-03 18:32');
Insert into Ballot(mid,oid,ballotPickupTime) Values(2,1,'2009-10-02 23:41');
Insert into Ballot(mid,oid,ballotPickupTime) Values(3,1,'2009-10-03 14:53');
Insert into Ballot(mid,oid,ballotPickupTime) Values(4,1,'2009-10-03 06:32');
Insert into Ballot(mid,oid,ballotPickupTime) Values(6,1,'2009-10-02 23:45');
Insert into Ballot(mid,oid,ballotPickupTime) Values(7,1,'2009-10-03 04:02');
```

Table: **Ballot**

n	n	c
mid	**oid**	**ballotPickupTime**
1	1	2009-10-03 18:32
2	1	2009-10-02 23:41
3	1	2009-10-03 14:53
4	1	2009-10-03 06:32
6	1	2009-10-02 23:45
7	1	2009-10-03 04:02

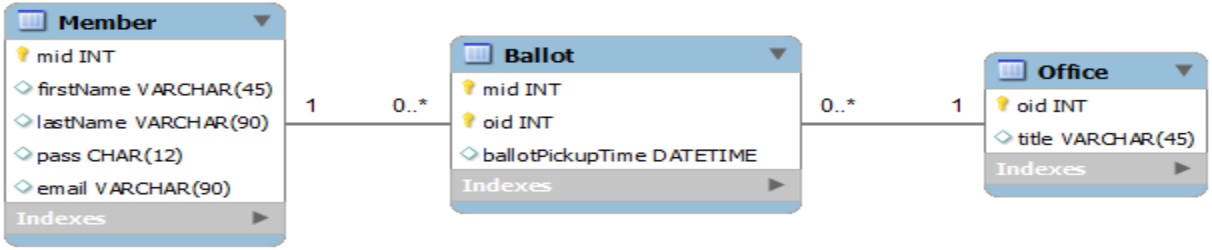


Table: **Member**

n	c	c	c	c
mid	**firstName**	**lastName**	**pass**	**email**
1	Ann	Anson	aa01119	aanson@misor.org
2	Ben	Jamin	bj11122	bjamin@misor.org
3	Cir	Cus	cc22211	ccus@misor.org
4	Don	Donski	dd00413	ddonski@misor.org
5	Eve	Lady	el98765	elady@misor.org
6	Fin	End	fe00011	fend@misor.org
7	Gin	Rum	gr12345	grum@misor.org

Table: **Ballot**

n	n	c
mid	**oid**	**ballotPickupTime**
1	1	2009-10-03 18:32
2	1	2009-10-02 23:41
3	1	2009-10-03 14:53
4	1	2009-10-03 06:32
6	1	2009-10-02 23:45
7	1	2009-10-03 04:02

Table: **Office**

n	c
oid	**title**
1	President
2	Vice-President
3	Treasurer
4	Editor
5	Web Manager

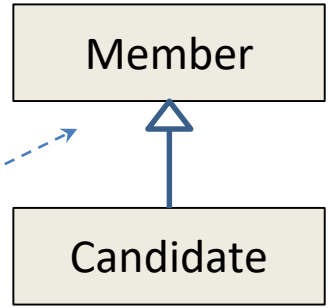
Looking at the relationships between the entities defined, using the primary and foreign keys, one can see that, for example, 'Ann' (pk: **mid**=1 in **Member**) picked her ballot for the office of 'President' (pk: **oid**=1 in **Office**) at **ballotPickupTime**='2009-10-03 18:32'. In table **Ballot**, the foreign keys, **mid** and **oid**, are set for this relationship to the values of the primary keys of 'Ann' and 'President', respectively. Jointly, in table **Ballot**, the keys **mid** and **oid** constitute the primary key.

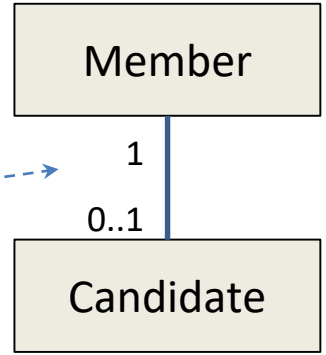
There is no requirement for a foreign key to have the same name as its related primary key. Nonetheless, having the same names may simplify documentation and query development as long as it is clear which are the primary keys and which are the foreign keys. Some of the **SQL** queries, involving more than one related tables can be simplified when the keys have the same names.

Notice that **MW** has generated the foreign key with names made of the entity names and their primary-key names (**Member_mid** and **Office_oid**) and we changed them to **mid** and **oid**, respectively.

In this instruction, all primary keys and their corresponding foreign keys will have the same name. Thus, the primary key, **mid**, of the **Member** table will propagate to other entities as a foreign key with the same name (**mid**). The primary key, **oid**, of the **Office** table, will also have the same name for all related foreign keys. It will make **SQL** queries look more slick and compact.

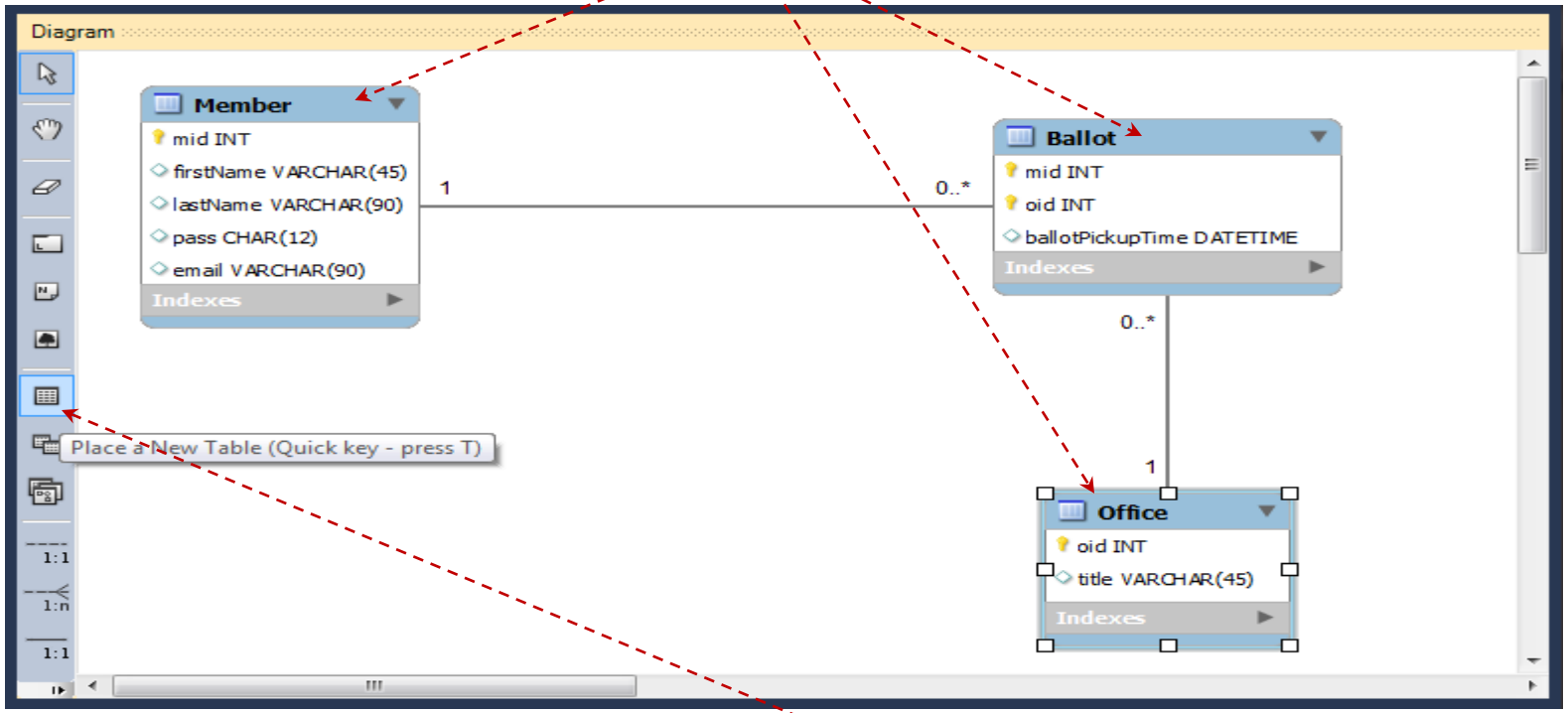
The model developed so far does not include important individuals: candidates who are running for offices. Each candidate must be a member (an instance of entity **Member**) but only some of the members are candidates (instances of entity **Candidate**). The relationship between **Candidate** and **Member** is hierarchical. **Candidates** inherit all the attributes from related members. Such a relationship is modeled in **UML** as **specialization** (a candidate is a specialization of a member) also known as a **Super-type – Sub-type** association*. **MySQL Workbench** does not support such a notation. However, it supports this type of the relationship as an **One-to-One** relationship with optional participation of the **subtype** entity (here **Candidate**).





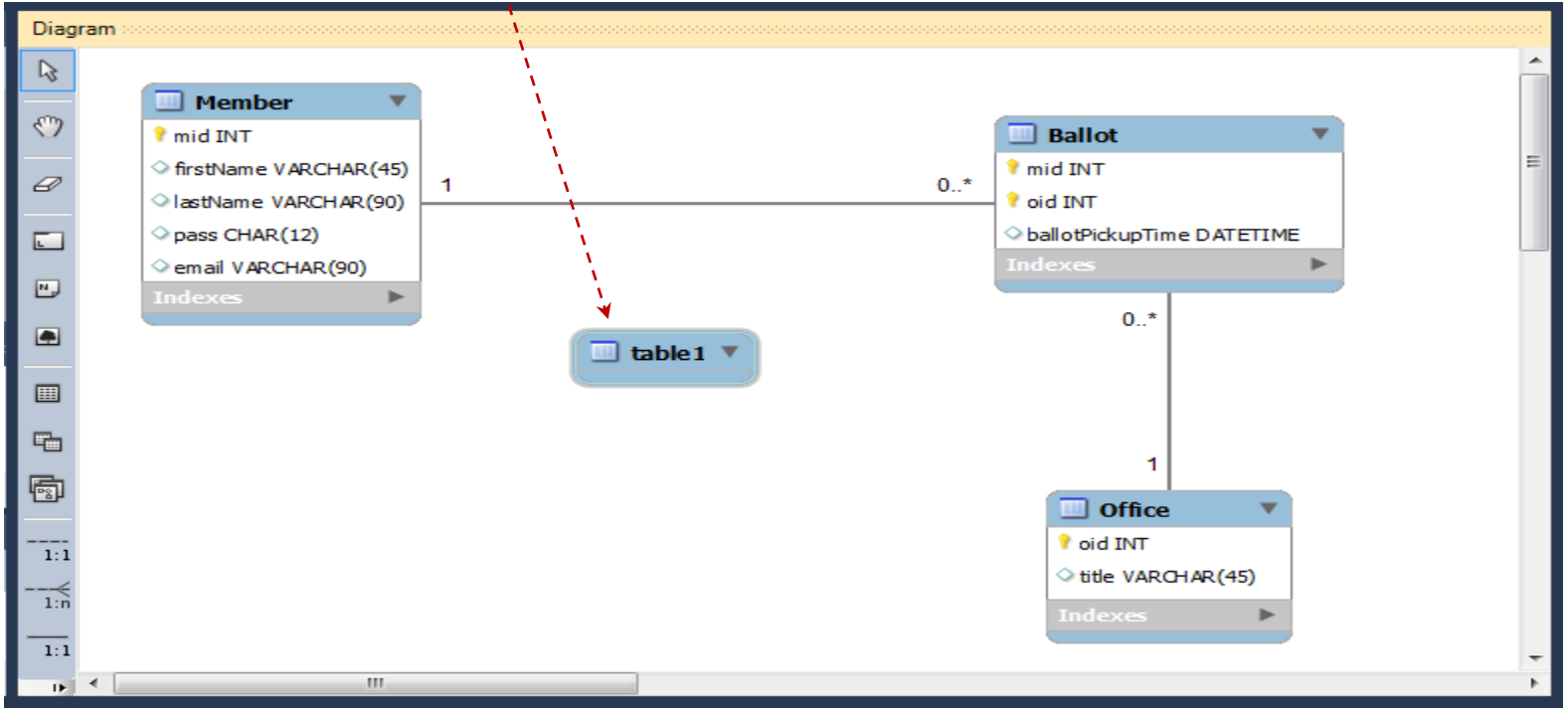
* In an object-oriented language , **Member** would be a super-class and **Candidate—**a sub-class. **Java** would say that class **Candidate** extends class **Member**.

Since there will be more entities and relationships in the model, why don't we ***rearrange*** the entities approximately as shown below.
It can be done by simply ***dragging*** the **entities** around.

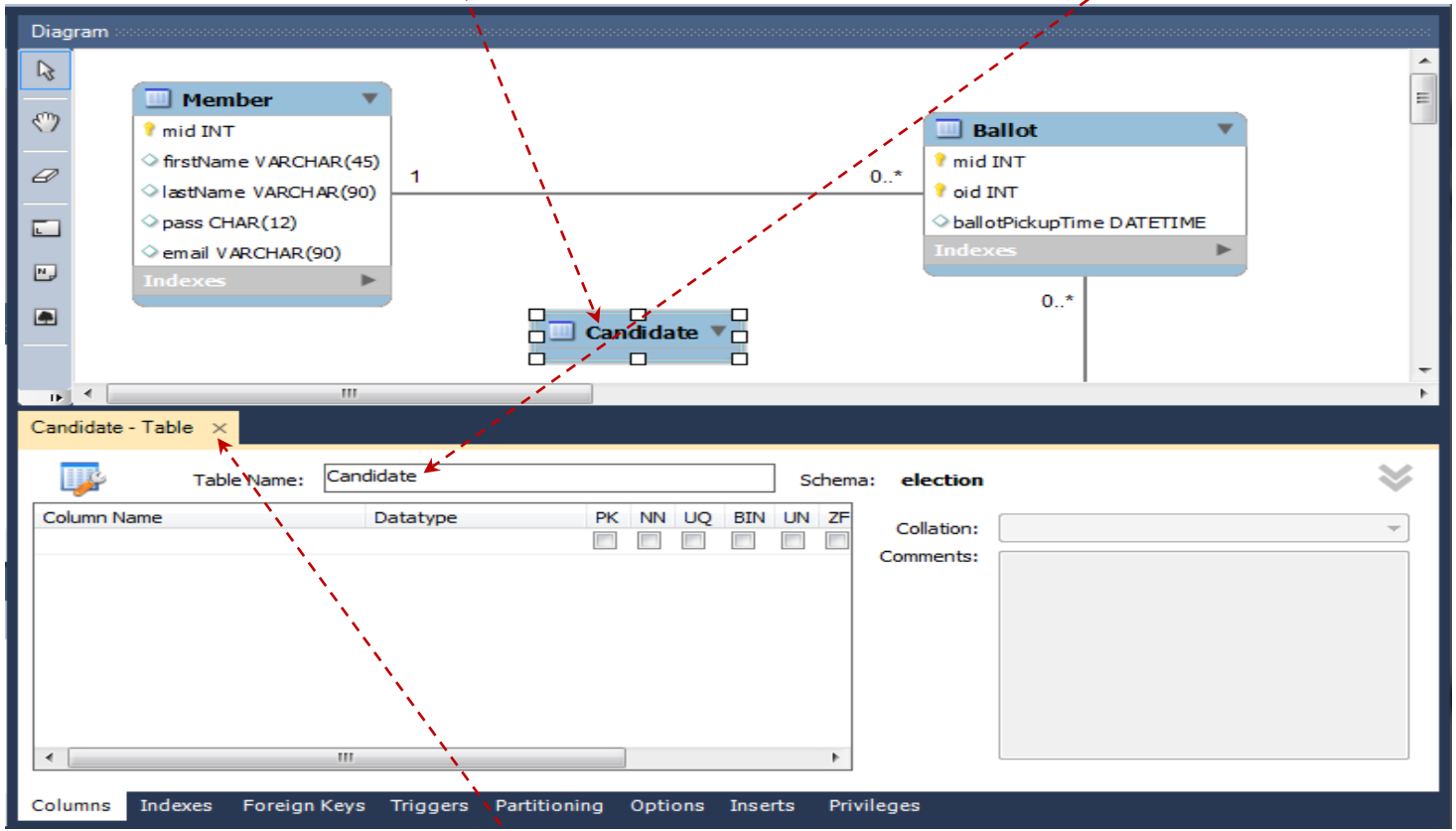


Now, ***click*** the **New Table** icon.

Click the **diagram canvas** to add a table.



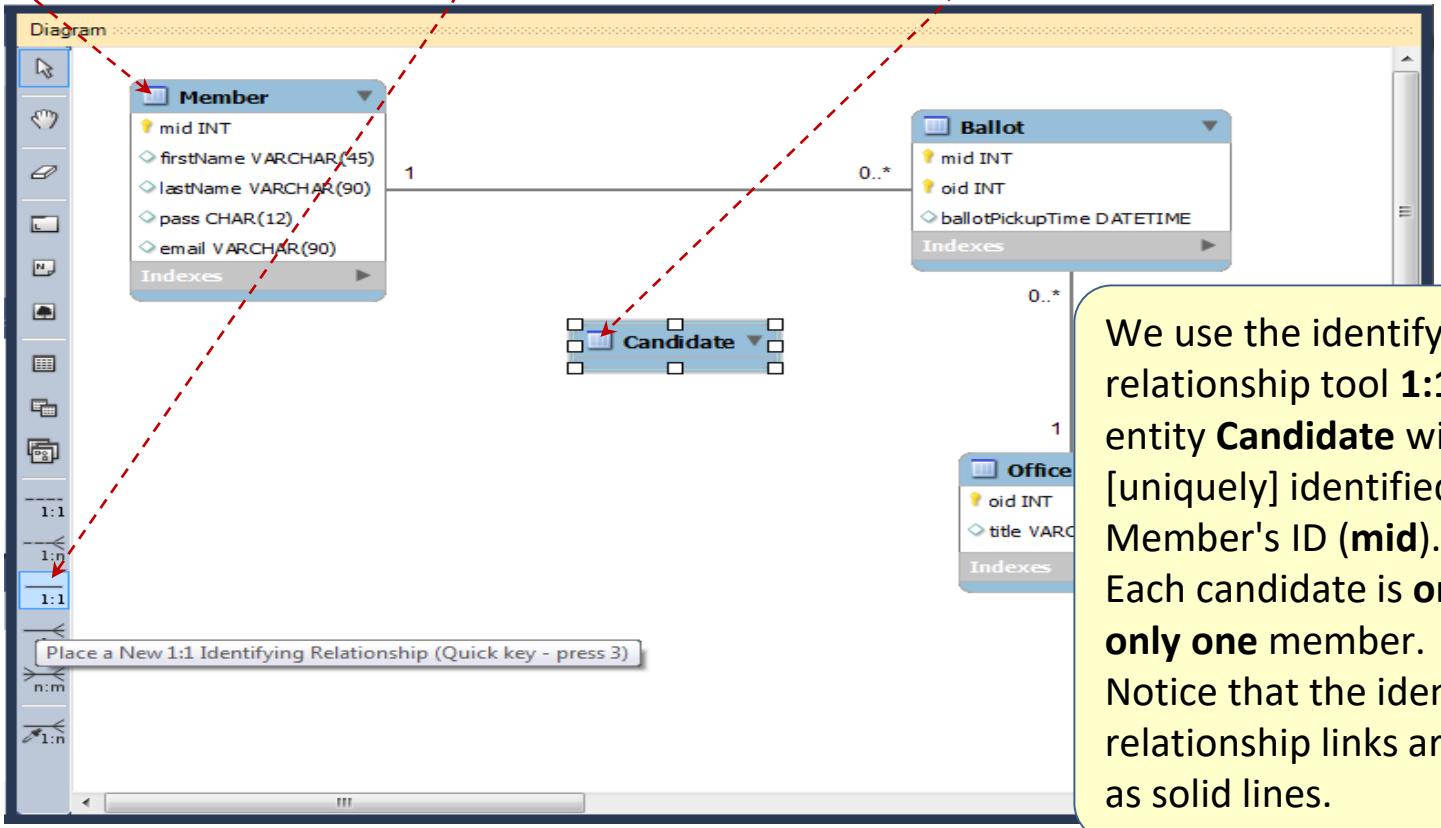
Double-click the **new table** and ***change*** its name to **Candidate**.



When done, ***close*** the **Candidate – Table** panel.

Connect entity **Candidate** with entity **Member** using the **1:1 Identifying Relationship** tool.

More specifically, first ***click*** the **1:1 tool**, next ***click*** table **Candidate** and finally ***click*** table **Member**.



We use the identifying relationship tool **1:1** since entity **Candidate** will be [uniquely] identified by the Member's ID (**mid**).
Each candidate is **one and only one** member.
Notice that the identifying relationship links are shown as solid lines.

To *change* the name of the inherited table, *double-click* entity **Candidate** and change column-name **Member_mid** to **mid**.

Diagram

Member
mid INT
firstName VARCHAR(45)
lastName VARCHAR(90)
pass CHAR(12)
email VARCHAR(90)
Indexes

Ballot
mid INT
pid INT
ballotPickupTime DATETIME
Indexes

Candidate
mid INT
Indexes

Candidate - Table

Table Name: Candidate
Schema: election

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF
mid	INT	☑	☑	☐	☐	☐	☐

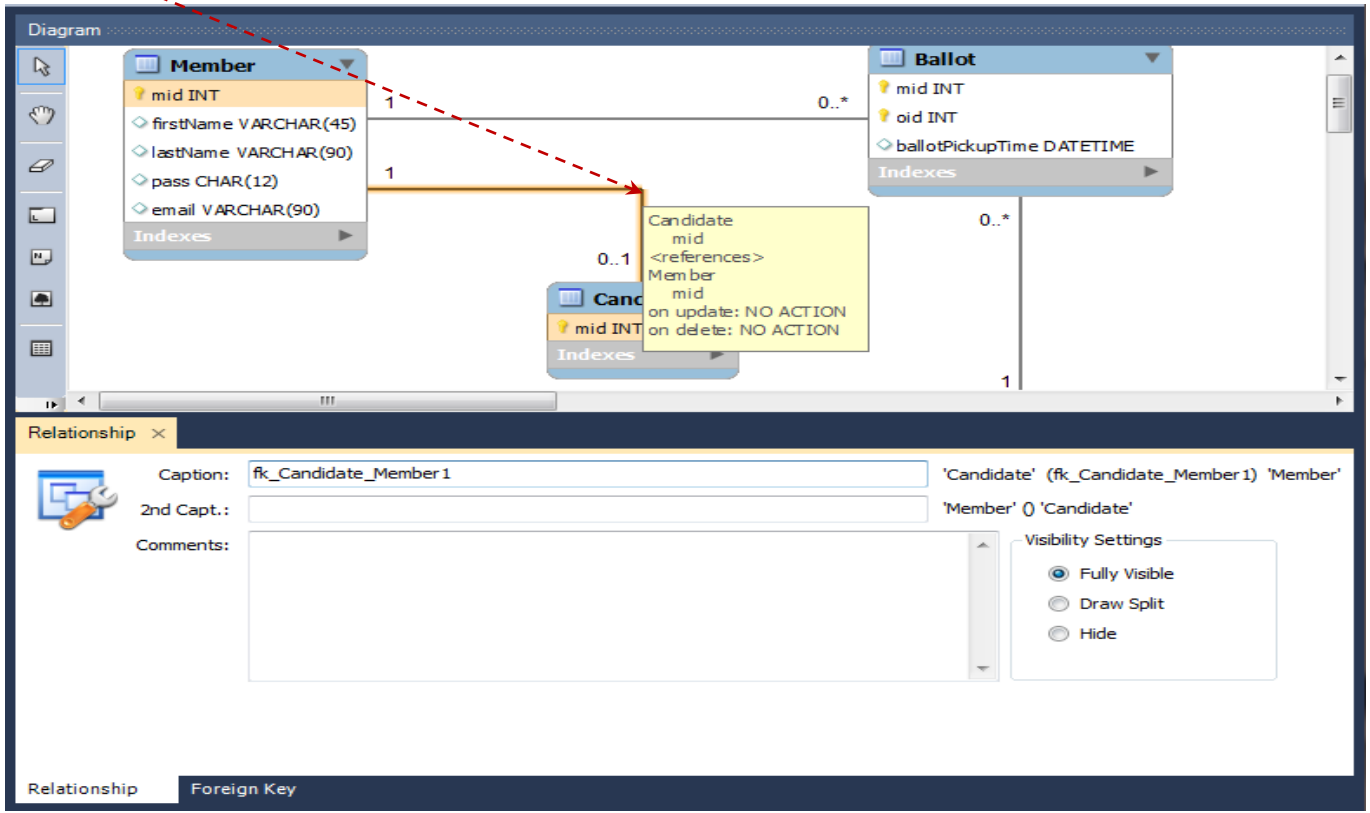
Collation: Table Default
Comments:

Columns | Indexes | Foreign Keys | Triggers | Partitioning

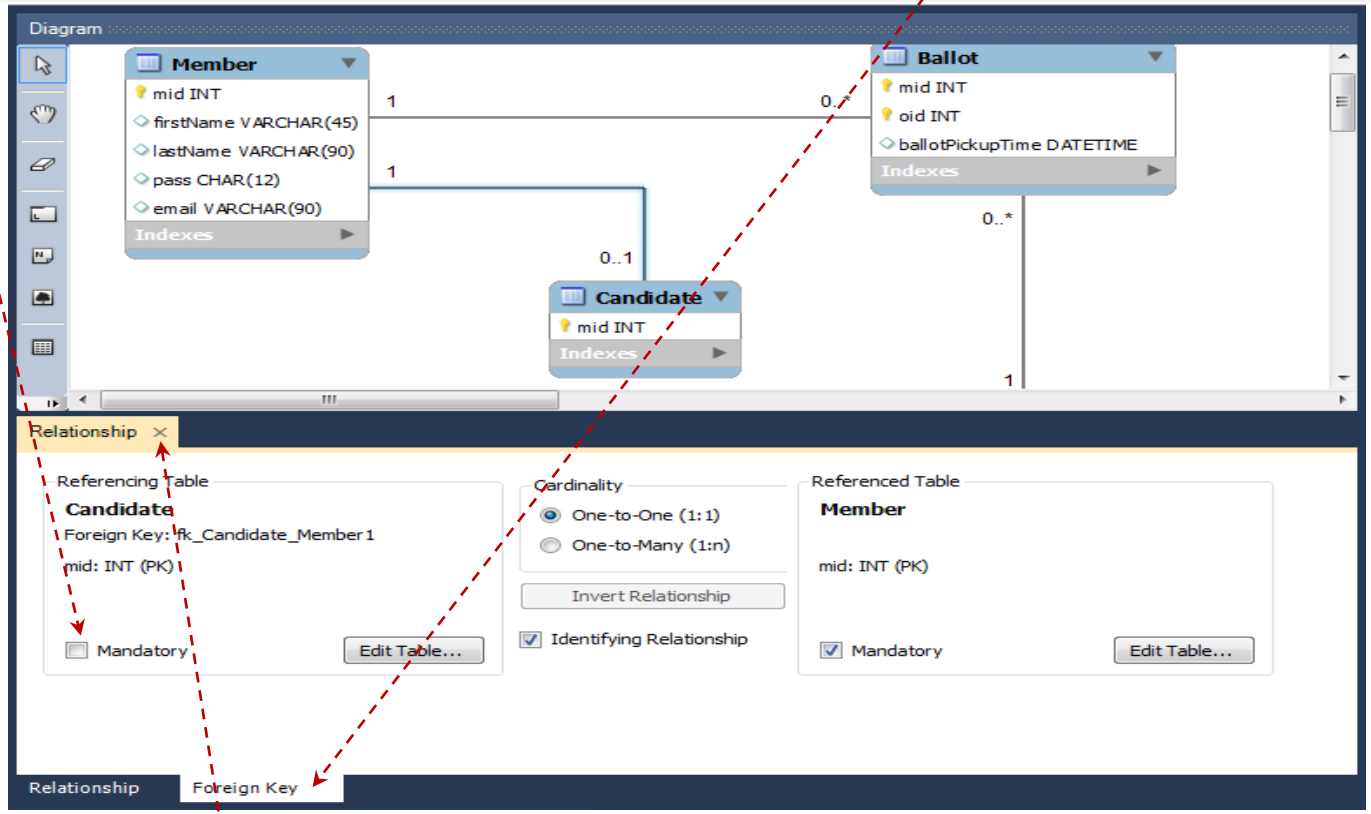
Notice that the inherited attribute, **mid**, serves in entity **Candidate** as both the **primary key** and the **foreign key**.

When done, *close* the **Candidate – Table** panel.

To *change* the cardinality constraint at the **Candidate** entity, first *double-click* the **Member - Candidate** link.



Select the **Foreign Key** tab and *uncheck* option **Mandatory** in the **Candidate** panel.



When done, *close* the **Relationship** panel.

A **One-to-Many** relationship is directional. When connecting the related entities, **MW** requires that the entity on the **Many** side be selected (clicked on) prior to selection of the entity at the **One** side. The latter maps its primary key to the foreign key of the former.

The relationship we are about to create is:

Office (1) — *is-being-run-for-by* — (1..*) **Candidate**

(In order for an instance of office, e.g. `'President'`, to be part of the election, there must be at least one candidate running for it.)

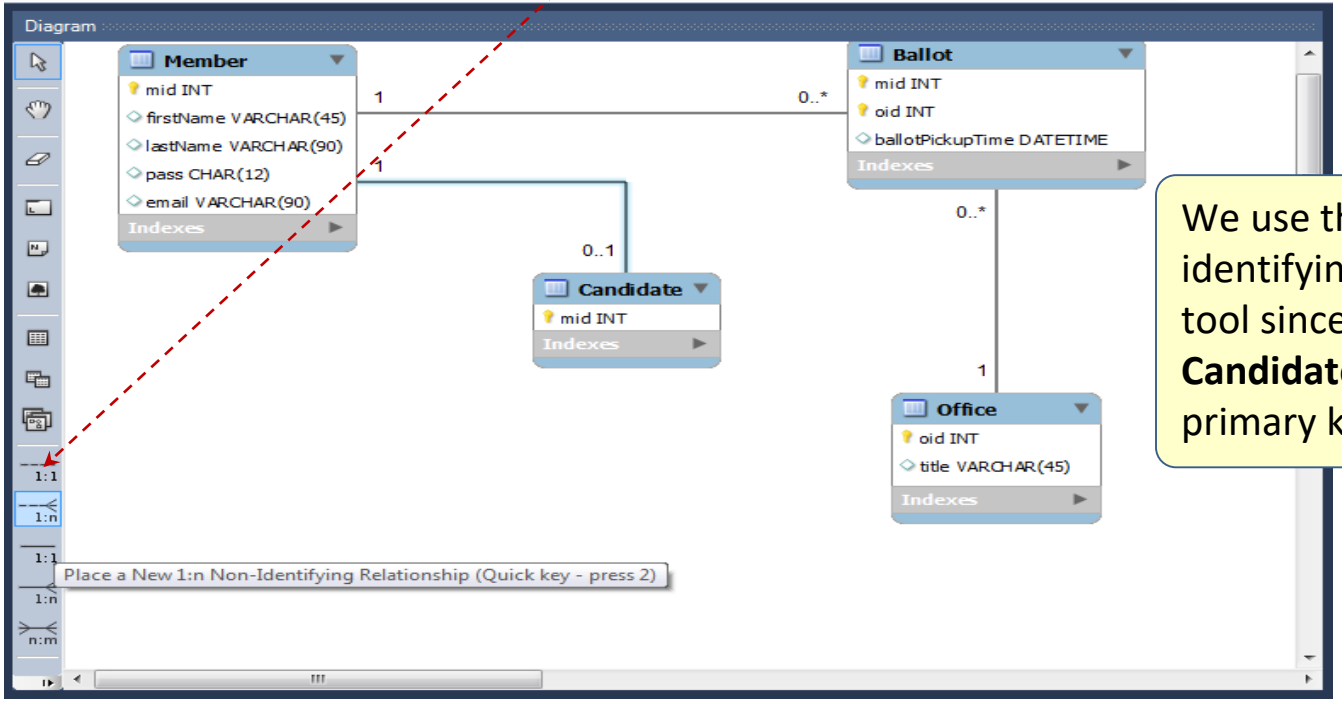
This relationship can also be defined as:

Candidate (1..*) — *runs-for* — (1) **Office**

Since an office may have many candidates, it is responsibility of the candidates to "know" which offices they are running for. Thus each instance of entity **Candidate** must include "information" about the office s/he is running for. This information is nothing else but the foreign key in the **Candidate** table pointing to the primary key in the **Office** table.

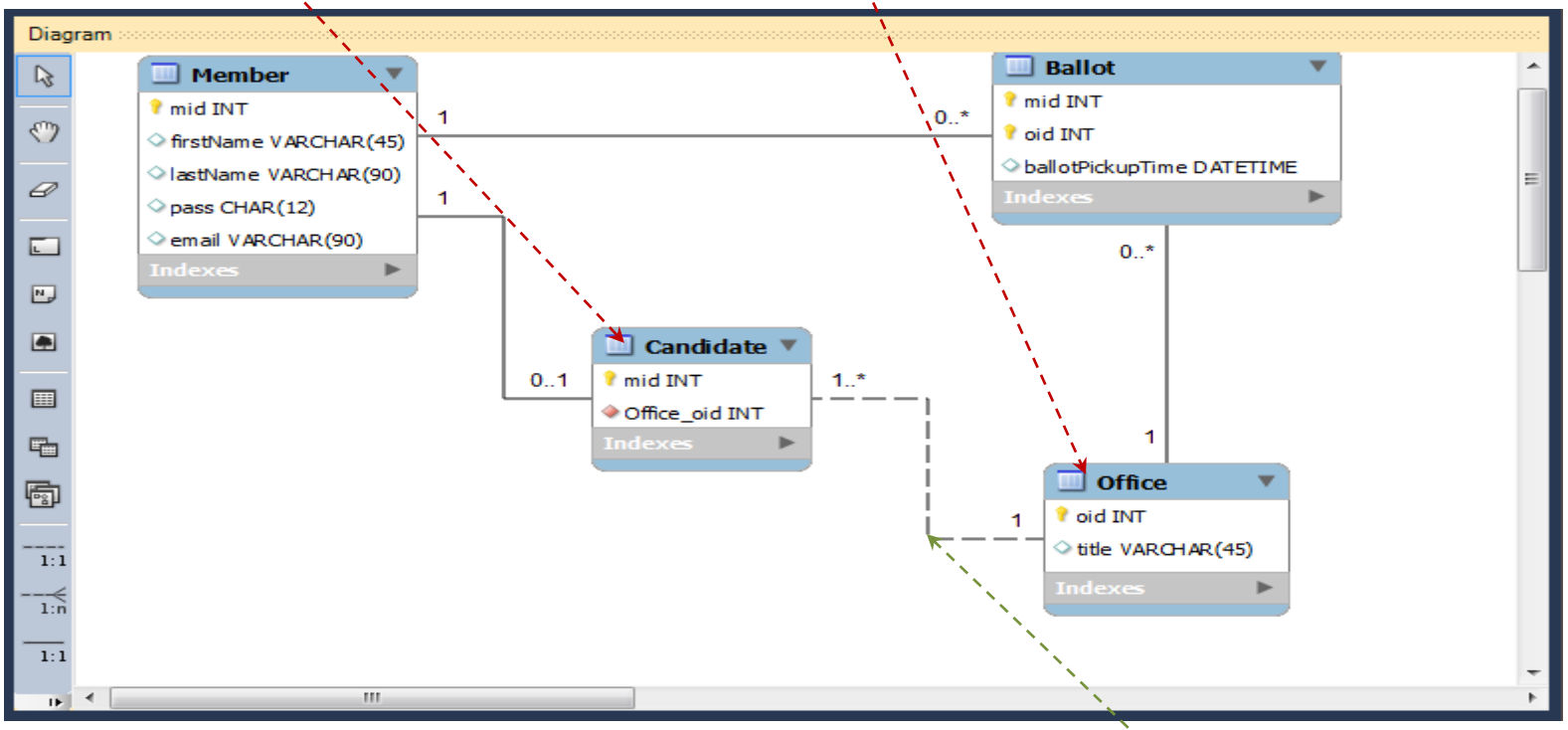
Now it is a good time to further specialize entity **Candidate**. What distinguishes a candidate from a member is that the former is [deterministically] running for an office. Since a candidate may run for only one office and an office may have many candidates, the appropriate relationship between **Office** and **Candidate** is a **One-To-Many (1:n) Non-identifying Relationship**.

Select (***click***) the **Non-identifying One-To-Many Relationship** tool.



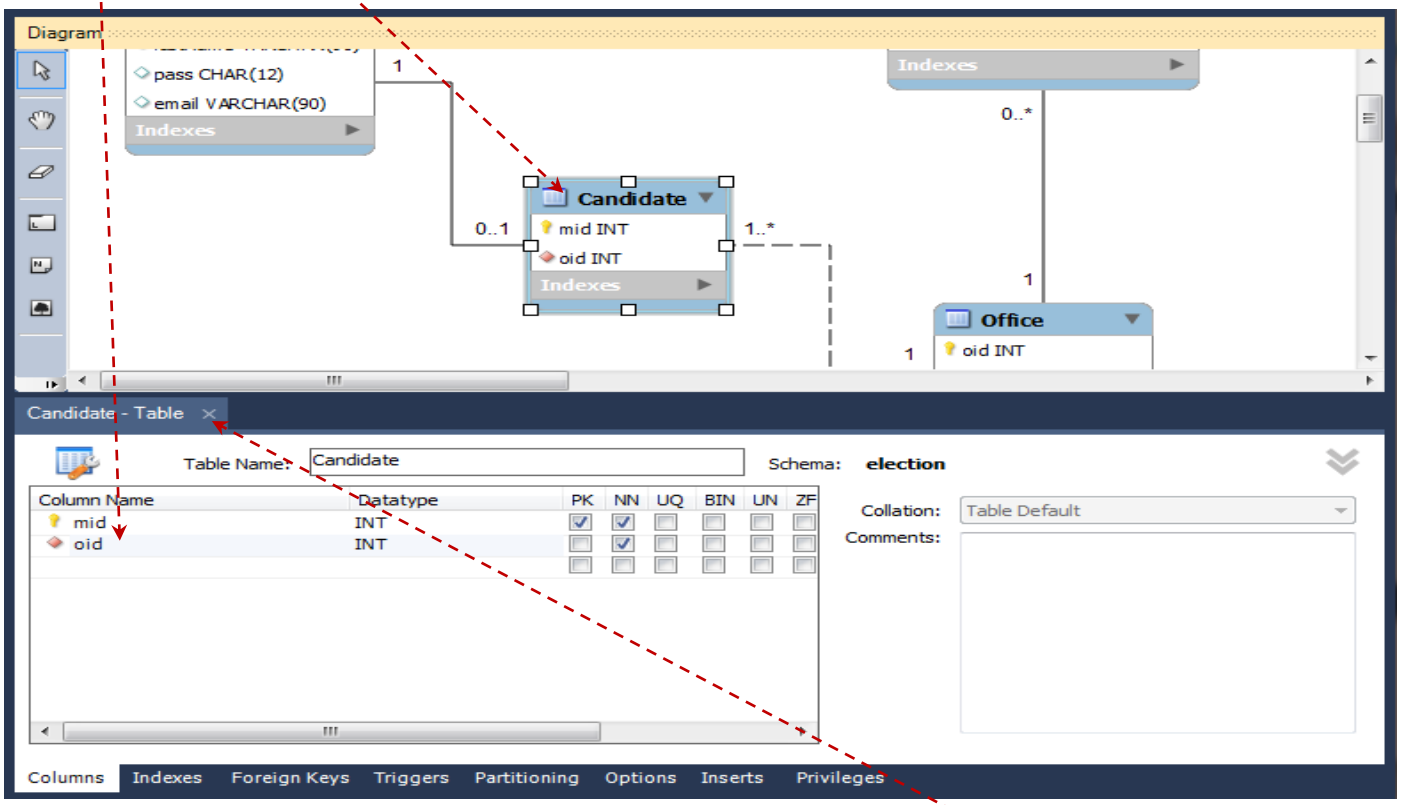
We use the non-identifying relationship tool since entity **Candidate** already has a primary key.

Click the **Candidate** table and then *click* the **Office** table.



MW connects entity **Office** with entity **Candidate** with a One-To-Many (**1 : 1..***) Non-identifying Relationship. Notice that this non-identifying relationship link is shown using a dashed line.

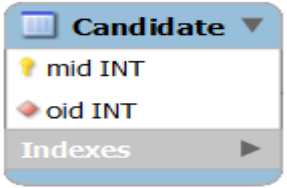
Double-click the **Candidate** table and then *change* column name **Office_oid** to **oid**.



When done, *close* the **Candidate – Table** panel.

The **Candidate** worksheet of the **election.xlsm** workbook shows three views of each of the **Candidate** entity: **UML**, **SQL** and **Excel Table**.

The `sqlInsert()` function transforms the rows of the table into **SQL-Insert** statements.

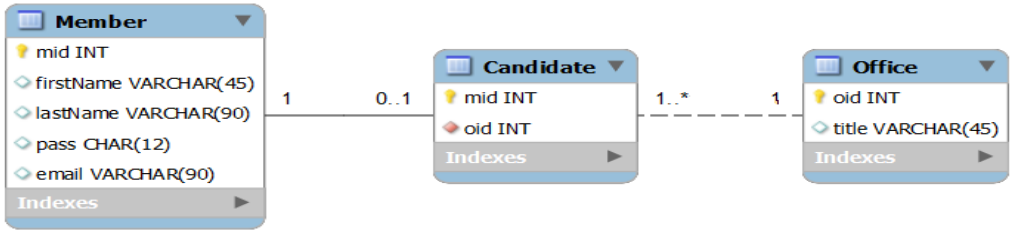



```
CREATE TABLE Candidate
(
    mid int primary key,
    oid int NOT NULL,
    FOREIGN KEY (mid) REFERENCES Member(mid),
    FOREIGN KEY (oid) REFERENCES Office(oid)
);
```

```
Insert into Candidate(mid,oid) Values(1,4);
Insert into Candidate(mid,oid) Values(3,1);
Insert into Candidate(mid,oid) Values(4,2);
Insert into Candidate(mid,oid) Values(7,4);
Insert into Candidate(mid,oid) Values(8,3);
Insert into Candidate(mid,oid) Values(9,3);
Insert into Candidate(mid,oid) Values(10,2);
```

Table: Candidate

n	n
mid	**oid**
1	4
3	1
4	2
7	4
8	3
9	3
10	2



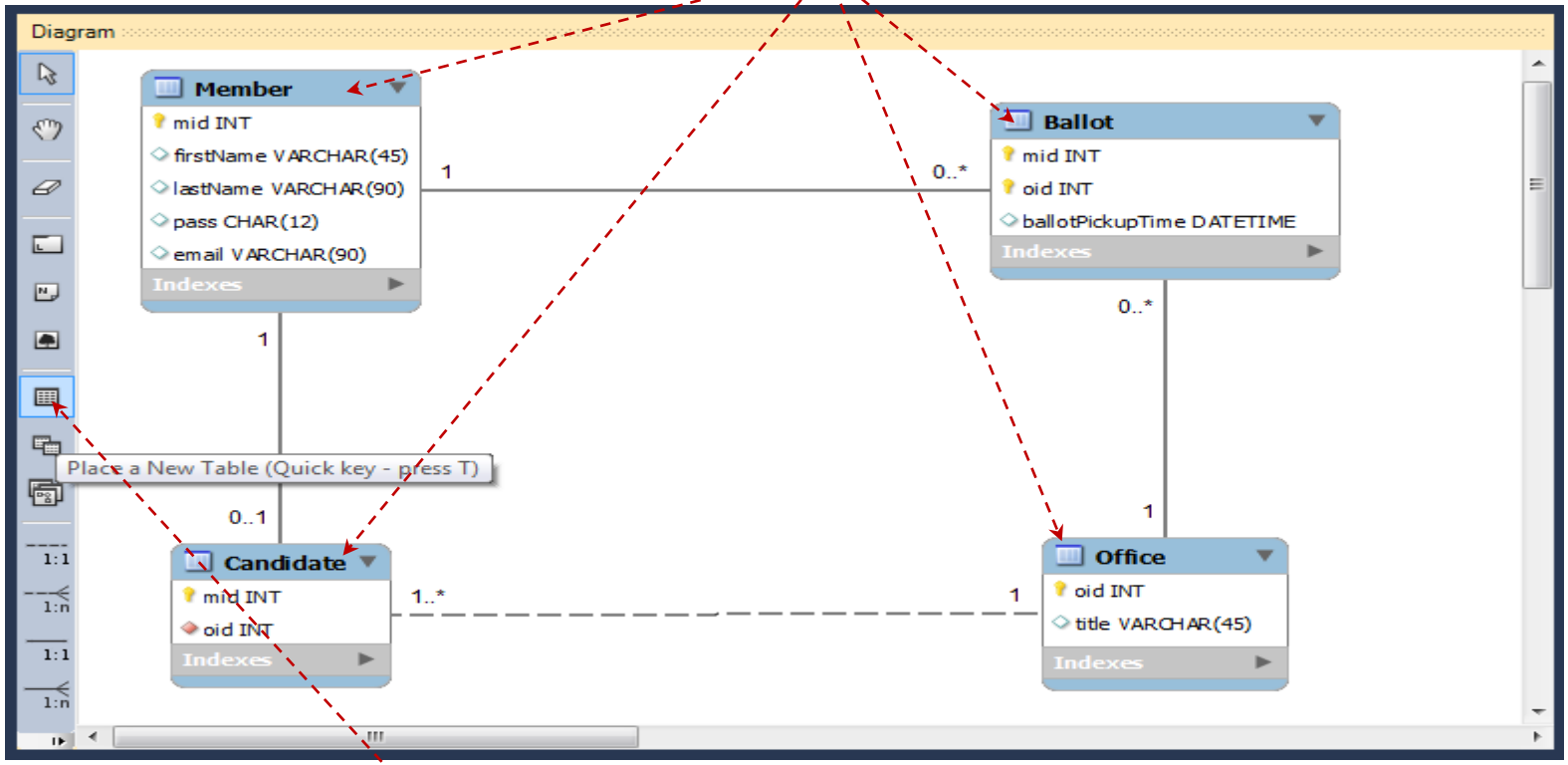
mid	firstName	lastName	pass	email
1	Ann	Anson	aa01119	aanson@misor.org
2	Ben	Jamin	bj11122	bjamin@misor.org
3	Cir	Cus	cc22211	ccus@misor.org
4	Don	Donski	dd00413	ddonski@misor.org
5	Eve	Lady	el98765	elady@misor.org
6	Fin	End	fe00011	fend@misor.org
7	Gin	Rum	gr12345	grum@misor.org
8	Hal	Bar	hb01011	hbar@misor.org
9	Ian	Yan	iy99900	iyan@misor.org
10	Jan	Osik	jo33322	josik@misor.org

mid	oid
1	4
3	1
4	2
7	4
8	3
9	3
10	2

oid	title
1	President
2	Vice-President
3	Treasurer
4	Editor
5	Web Manager

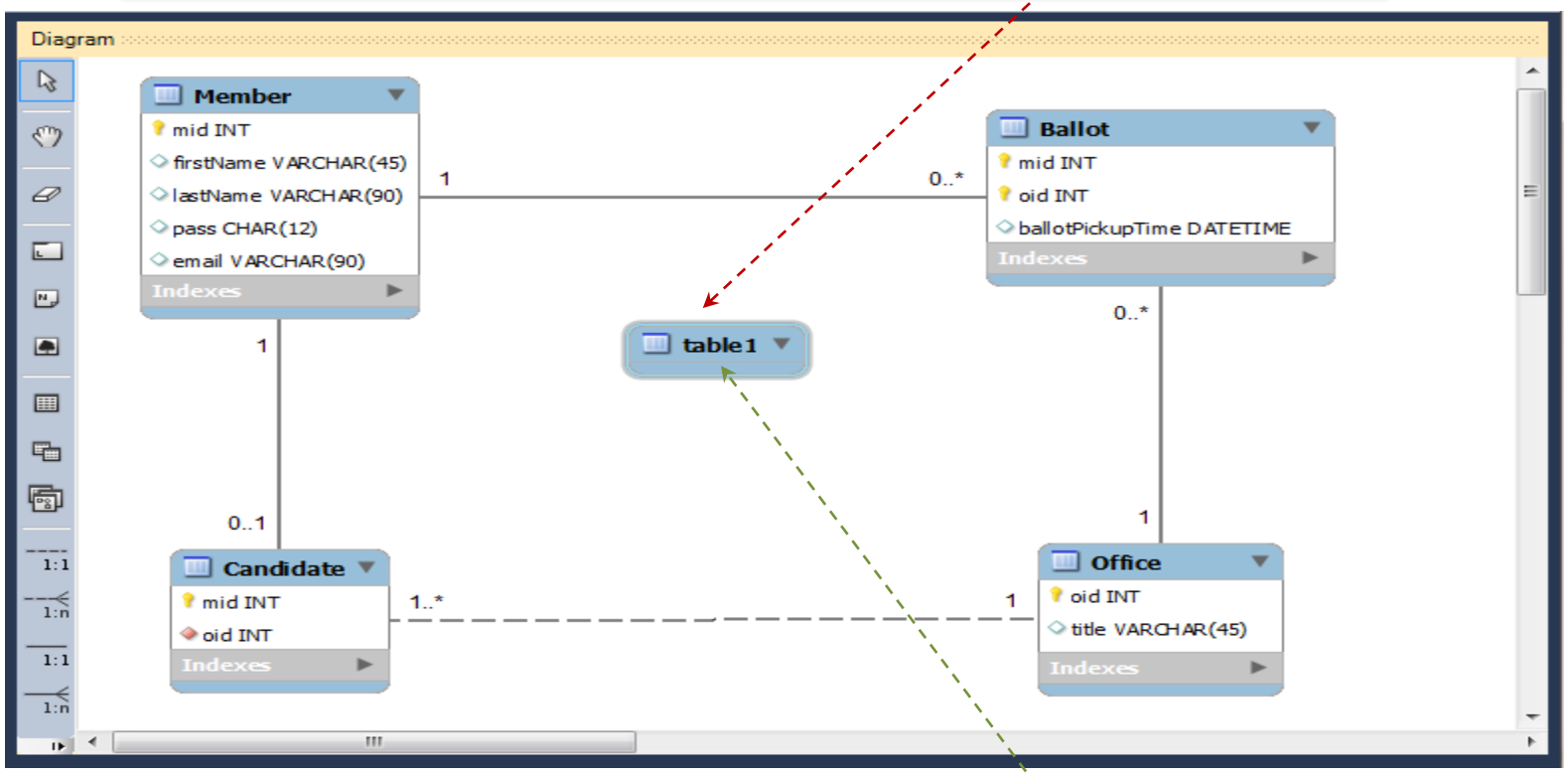
Looking at the relationships between the entities, defined by means of primary and foreign keys, one can see that, for example, `'Ann'` (pk: **mid**=1 in **Member**) is a candidate (pk: **mid**=1, having fk: **mid**=1 and **oid**=4 in **Candidate**) who is running for office of `'Editor'` (pk: **oid**=4) in **Office**. In table **Candidate**, the foreign keys, **mid** and **oid**, are set for this relationship to the values of the primary keys of `'Ann'` and `'Editor'`, respectively. In table **Candidate**, the key **mid** plays a dual role. It is both the primary and foreign key (resulting from an identifying relationship).

Since there will be one more entity and relationship in the model, why don't we ***rearrange*** the entities approximately as shown below.
It can be done by simply ***dragging*** the **entities** around.



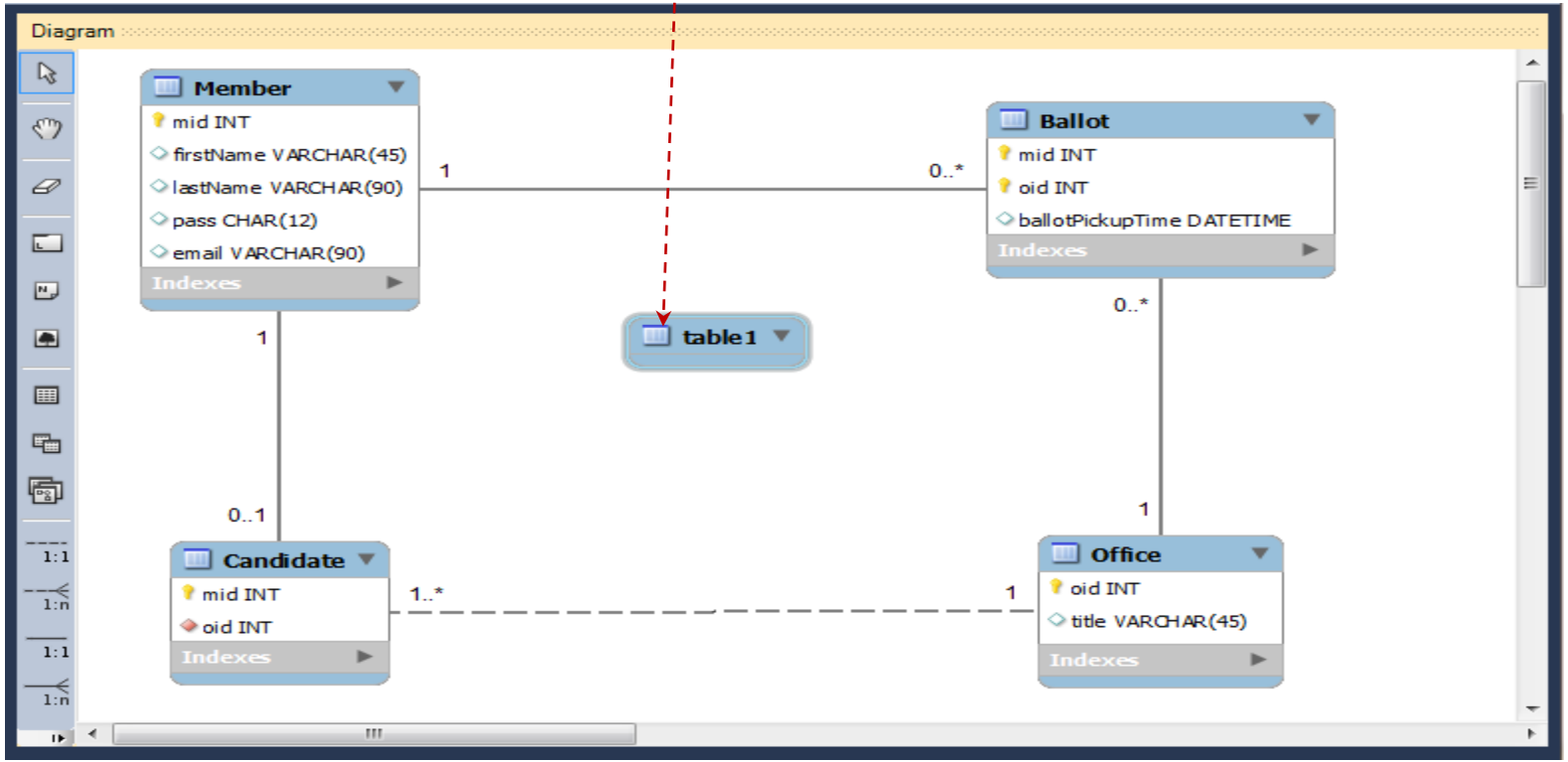
When done, ***click*** the **New Table** icon.

Click in the middle of the diagram to place a **new table** there.

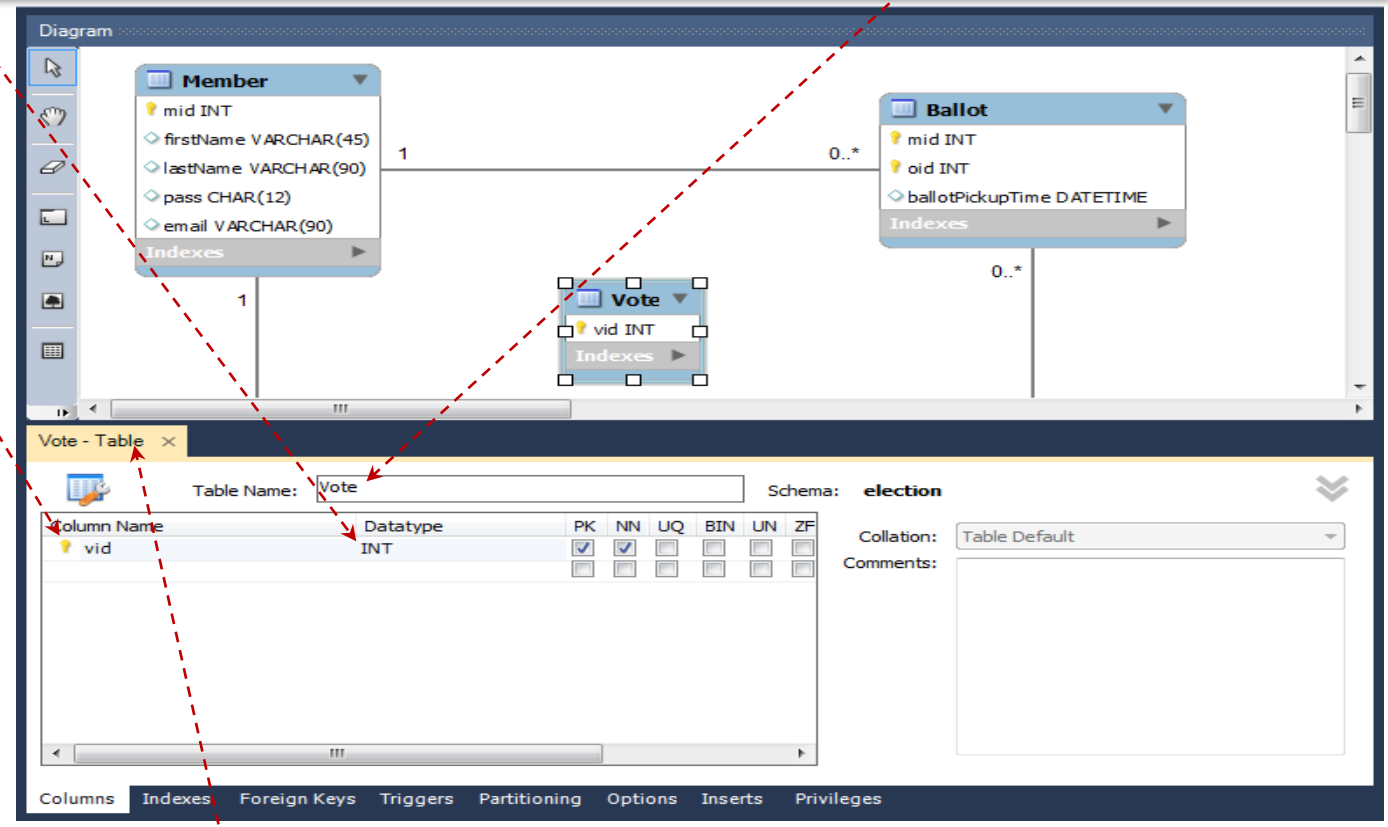


MW adds a new **entity** (**table1**) to the diagram.

Double-click the **new table**.

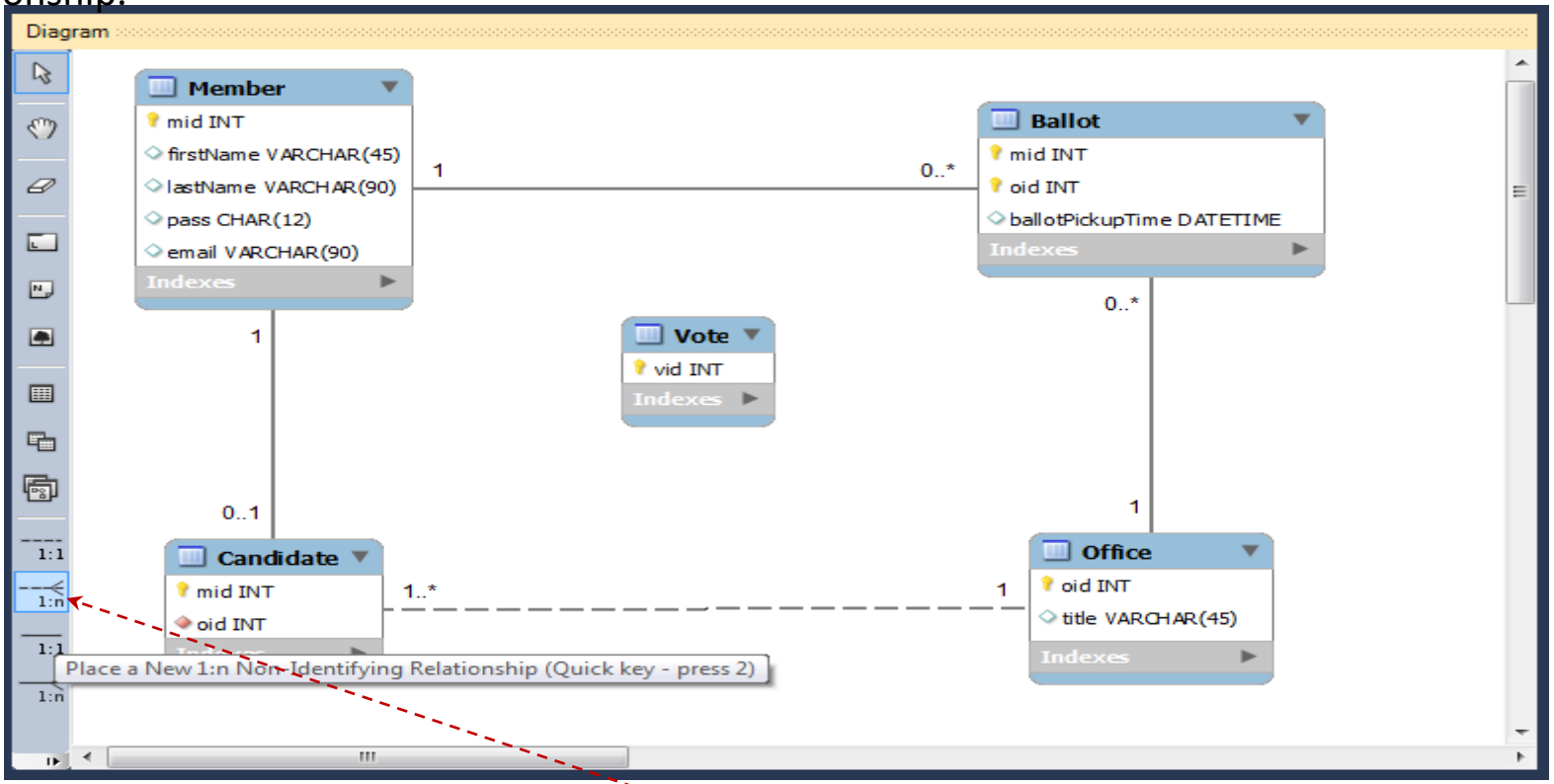


Change the name of this table to **Vote** and *add* column **vid** of type **INT** as a **primary key**.



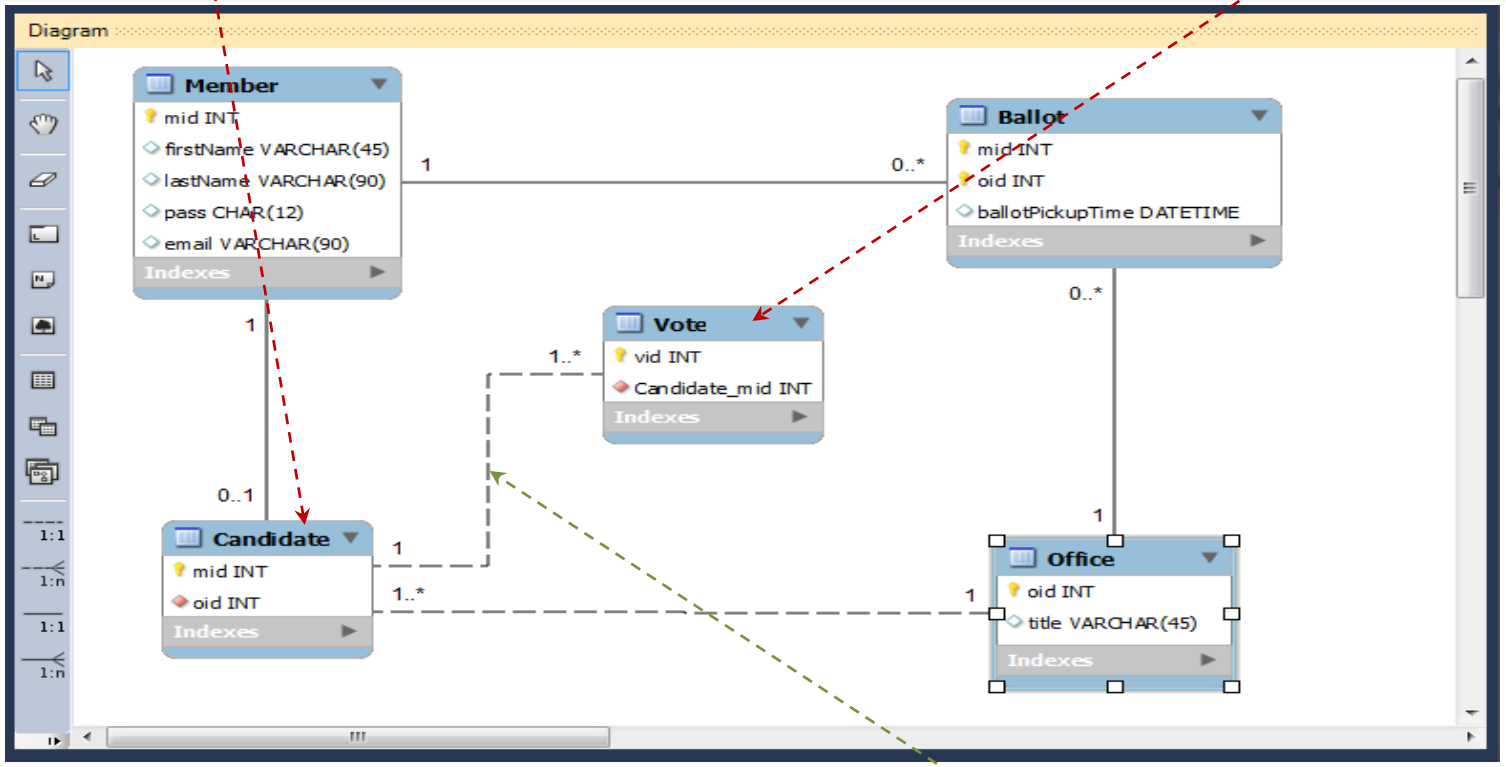
When done, *close* the **Vote – Table** panel.

Entity **Vote** serves here are a set of votes cast for the candidates. Since a candidate may get many votes and each vote is cast exactly for one candidate, **Candidate** – **Vote** is a **One-to-Many** relationship.



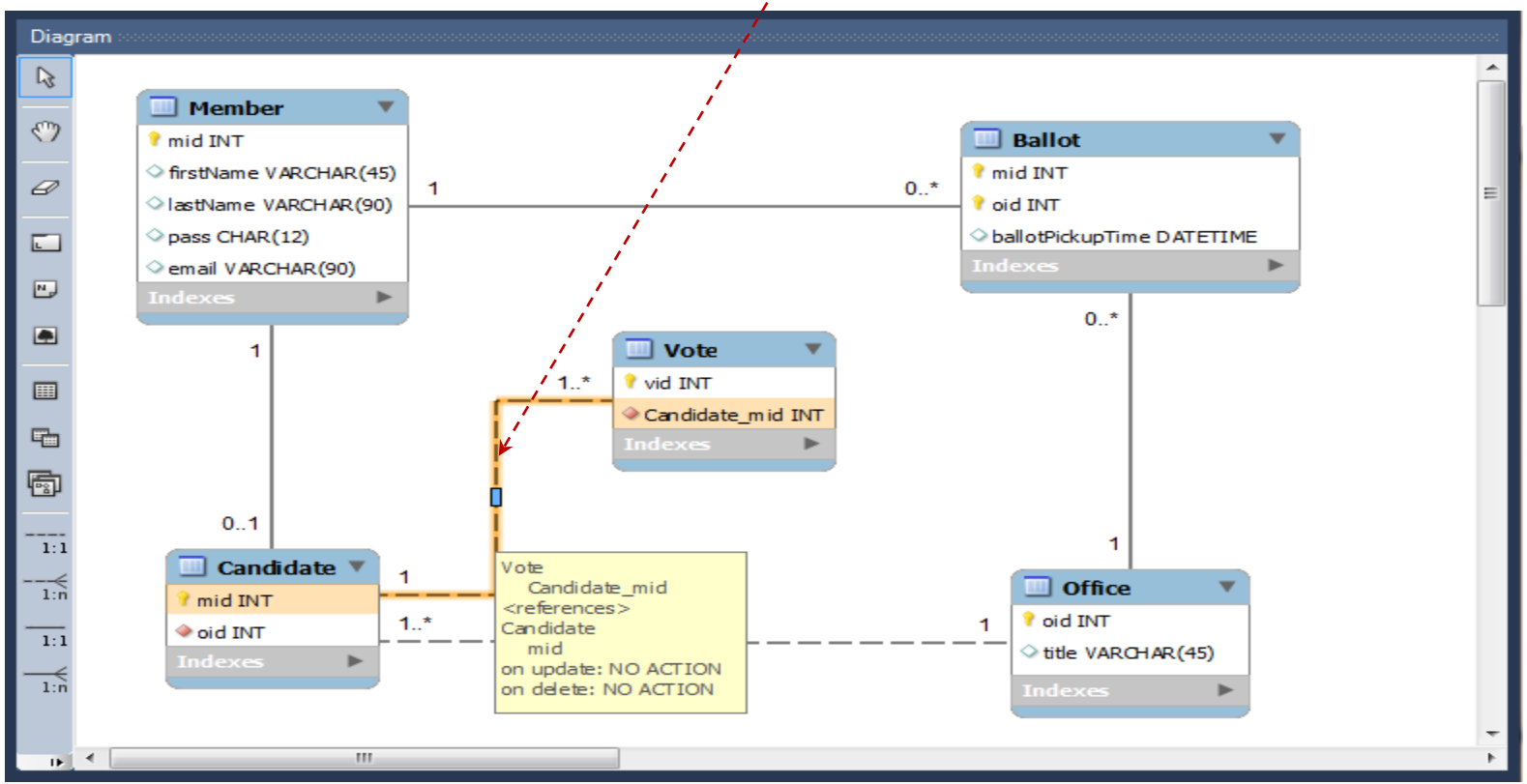
Select (***click***) the **1:n Non-Identifying Relationship** tool.

First *click* the **Vote** table and then *click* the **Candidate** table.

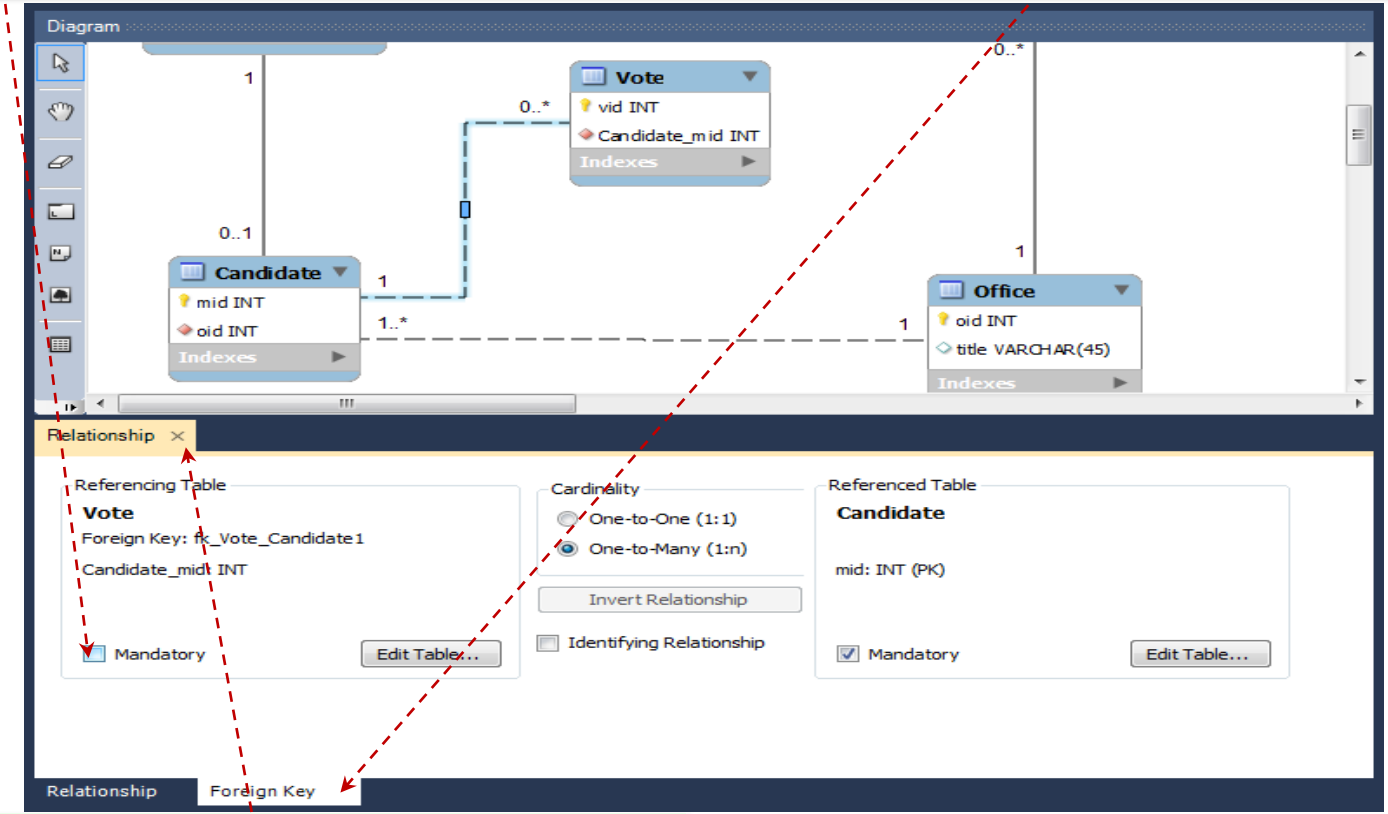


MW connects entity **Candidate** with entity **Vote** with a One-To-Many (**1 : 1..***) Non-identifying Relationship. This relationship needs to be "relaxed" as there may be candidates who will not receive any votes.

Double-click the **Candidate - Vote** relationship link.

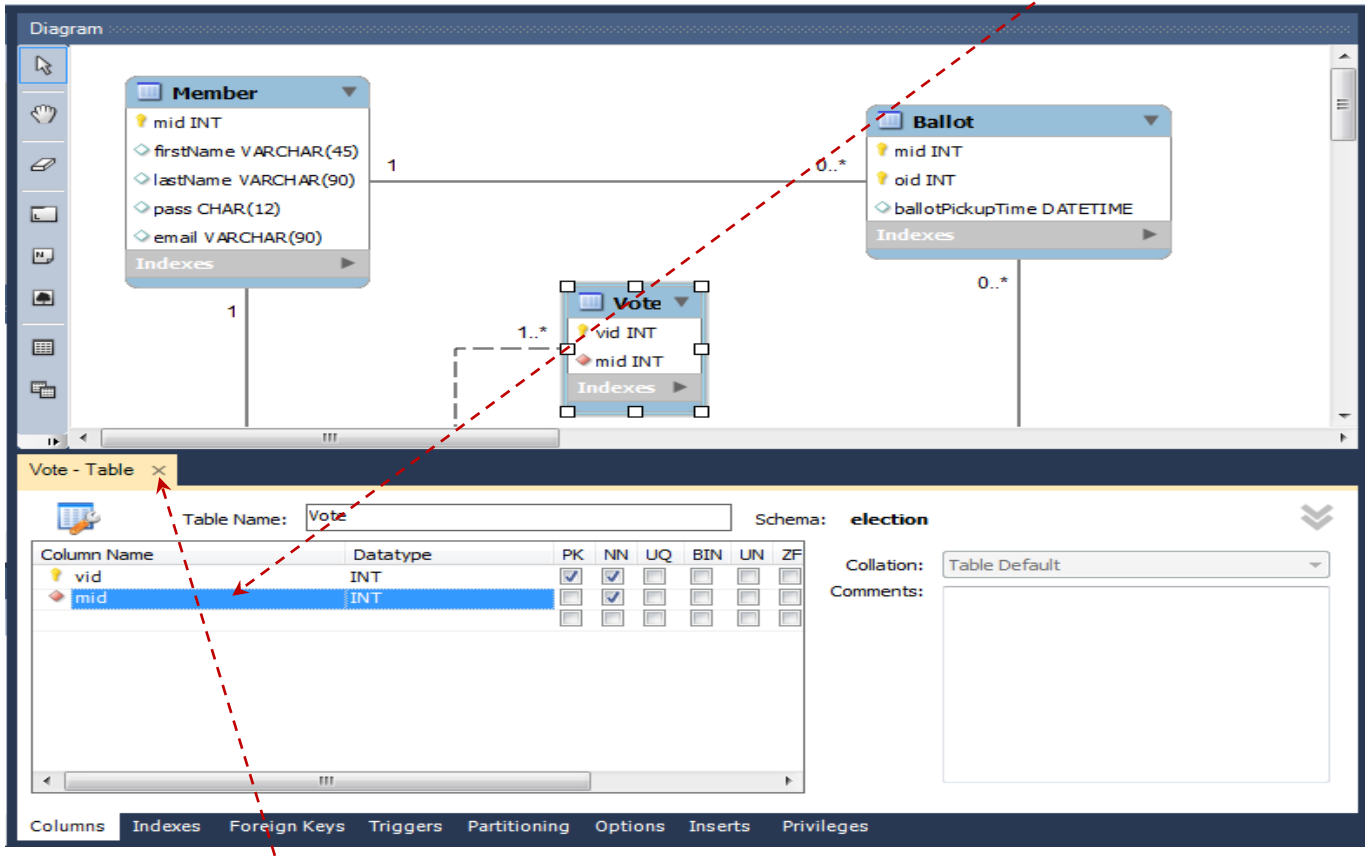


Select the **Foreign Key** tab and *uncheck* option **Mandatory** in the **Candidate** panel.



When done, *close* the **Relationship** panel.

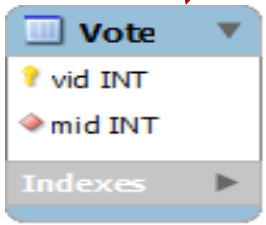
Change the name of column **Candidate_mid** to **mid**.



When done, *close* the **Vote – Table** panel.

The **Vote** worksheet of the **election.xlsm** workbook shows three views of each of the **Vote** entity: **UML**, **SQL** and **Excel Table**.

The `sqlInsert()` function transforms the rows of the table into **SQL-Insert** statements.




```
CREATE TABLE Vote
(
    id int primary key auto_increment,
    mid int NOT NULL,
    FOREIGN KEY (mid) REFERENCES
Candidate(mid)
);
```

```
Insert into Vote(vid,mid) Values(1,13);
Insert into Vote(vid,mid) Values(2,22);
Insert into Vote(vid,mid) Values(3,22);
Insert into Vote(vid,mid) Values(4,22);
Insert into Vote(vid,mid) Values(5,13);
Insert into Vote(vid,mid) Values(6,3);
Insert into Vote(vid,mid) Values(7,22);
Insert into Vote(vid,mid) Values(8,13);
Insert into Vote(vid,mid) Values(9,13);
Insert into Vote(vid,mid) Values(10,13);
```

Table:	Vote
n	n
vid	mid
1	13
2	22
3	22
4	22
5	13
6	3
7	22
8	13
9	13
10	13

Examining the relationships between the entities, defined by means of primary and foreign keys, one can see that, for example, candidate **13** (fk: **mid**=13 in **Candidate**) has received 5 votes (pk: **vid**=2, 3, 7, 11, 13, with fk: **mid**=13 in **Vote**).

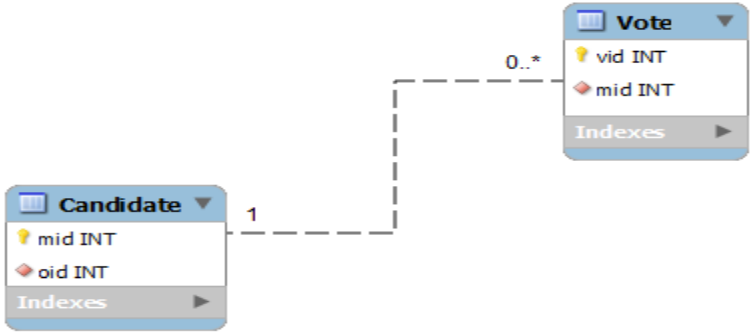


Table:	Candidate
n	n
mid	**oid**
1	4
3	1
4	2
7	4
8	3
9	3
10	2
12	3
13	1
15	2
18	5
20	4
21	4
22	1
23	4
24	2
25	4
33	5
37	5
40	5

Table:	Vote
n	n
vid	**cid**
1	22
2	13
3	13
4	22
5	22
6	3
7	13
8	22
9	3
10	22
11	13
12	22
13	13
14	3
15	3
16	3
17	22

The logical phase of the database design is done! It has enough details (tables, attributes, primary keys, foreign keys and cardinality constraints) so that it can be transformed into an executable **SQL** model. **MW** can do this transformation via its **Forward Engineer SQL** export facility (menu options):

File > **Export** > **Forward Engineer SQL Create Script ...**

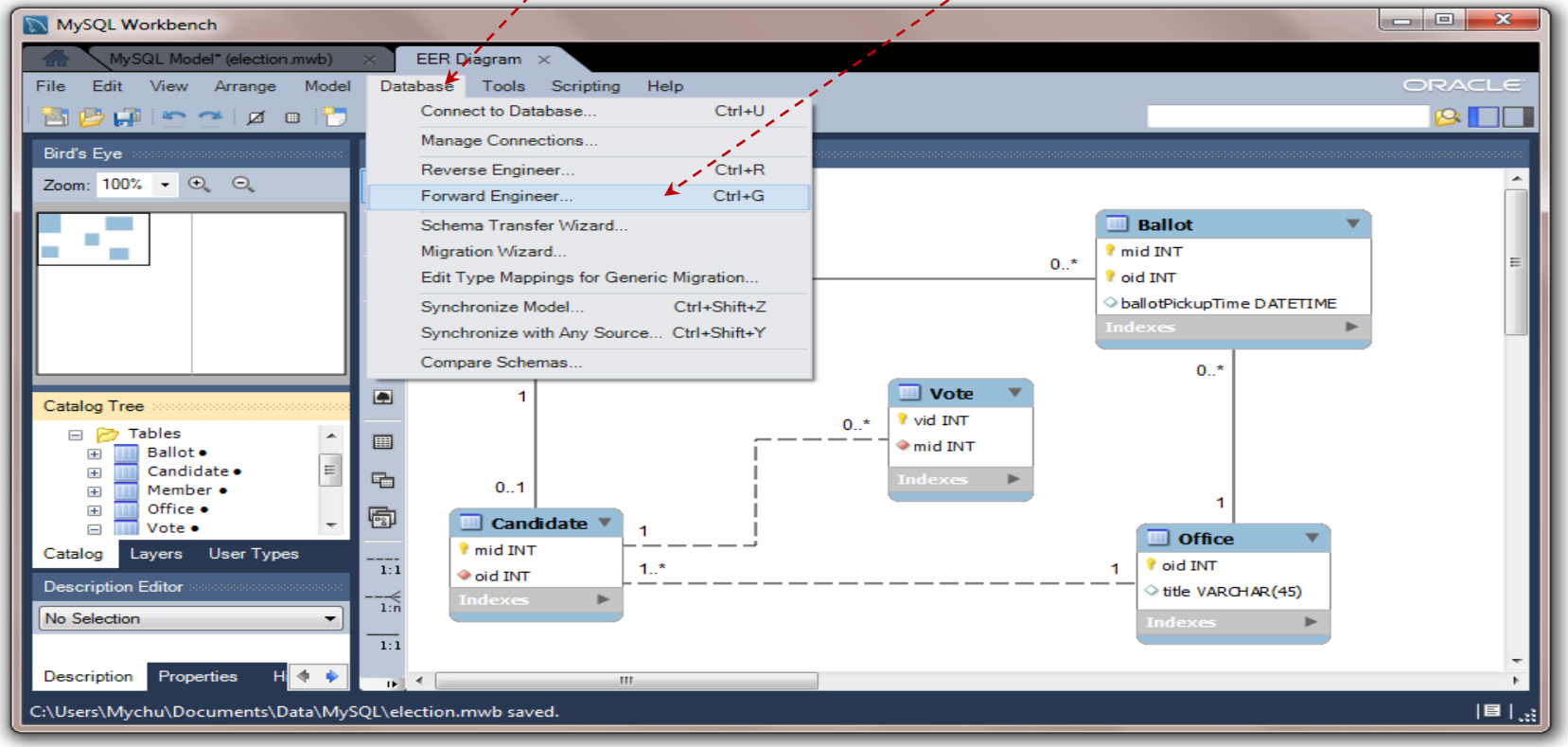
or

Database > **Forward Engineer ...**

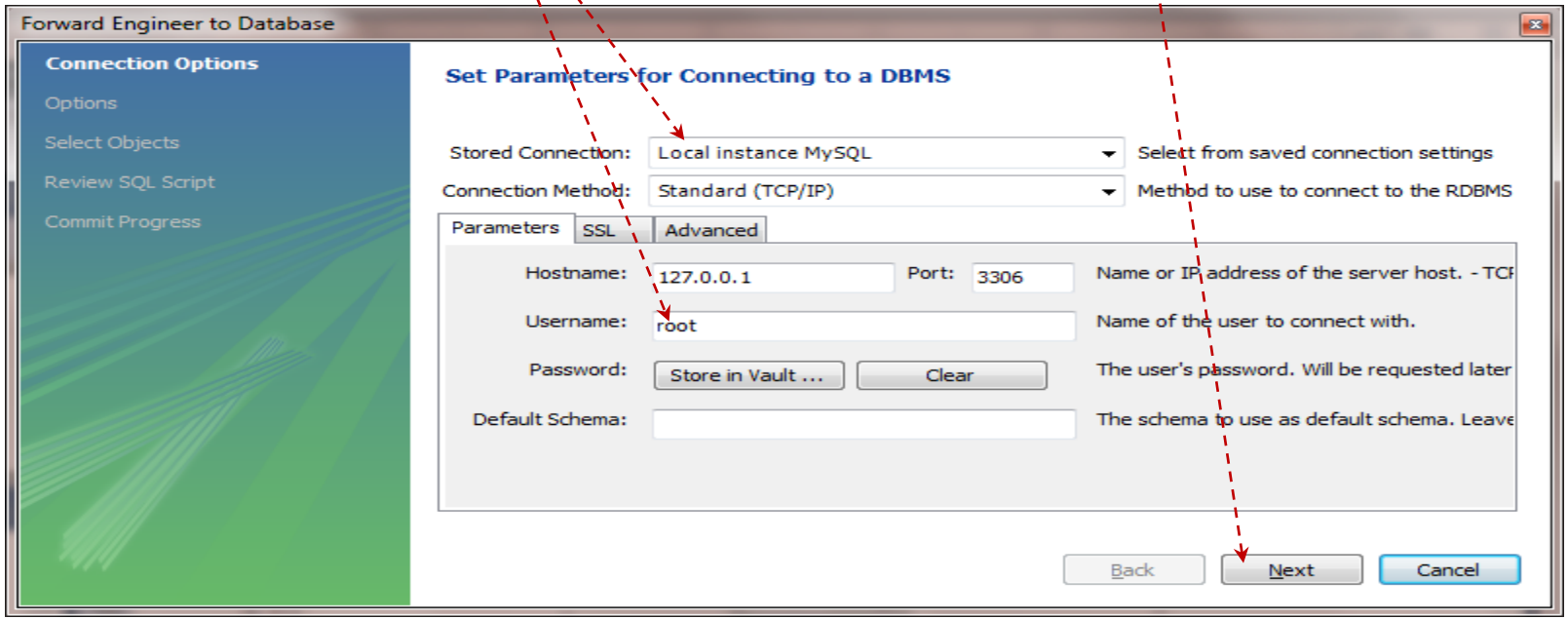
The following instruction shows how to do it using the latter command.

Make sure that your **EER** diagram is saved!

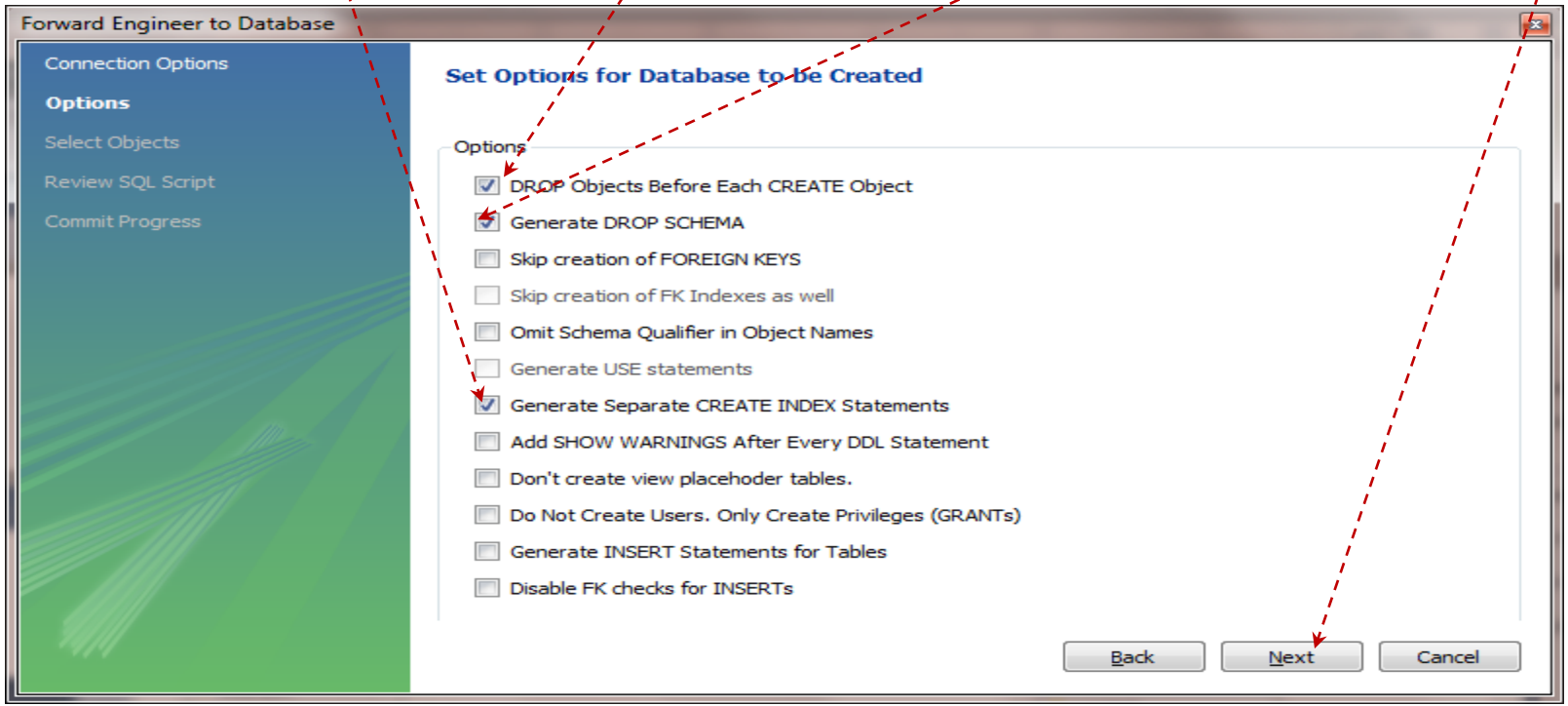
Select menu options **Database** followed by **Forward Engineer**.



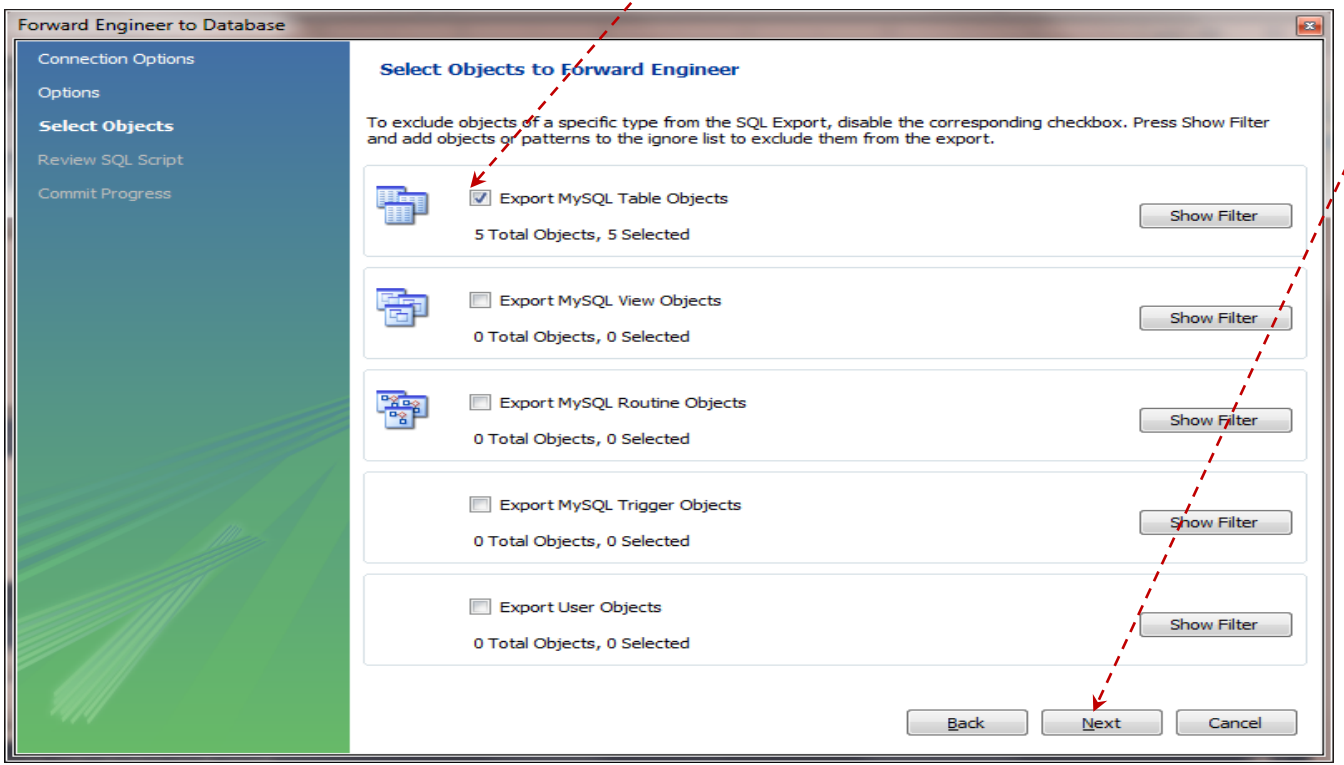
Use your **root** connection and *click* button **Next**.



Select (*check*) options **Drop Objects Before Each CREATE Object**, **Generate DROP SCHEMA**, and **Generate Separate CREATE INDEX Statements** and then *click* button **Next**.

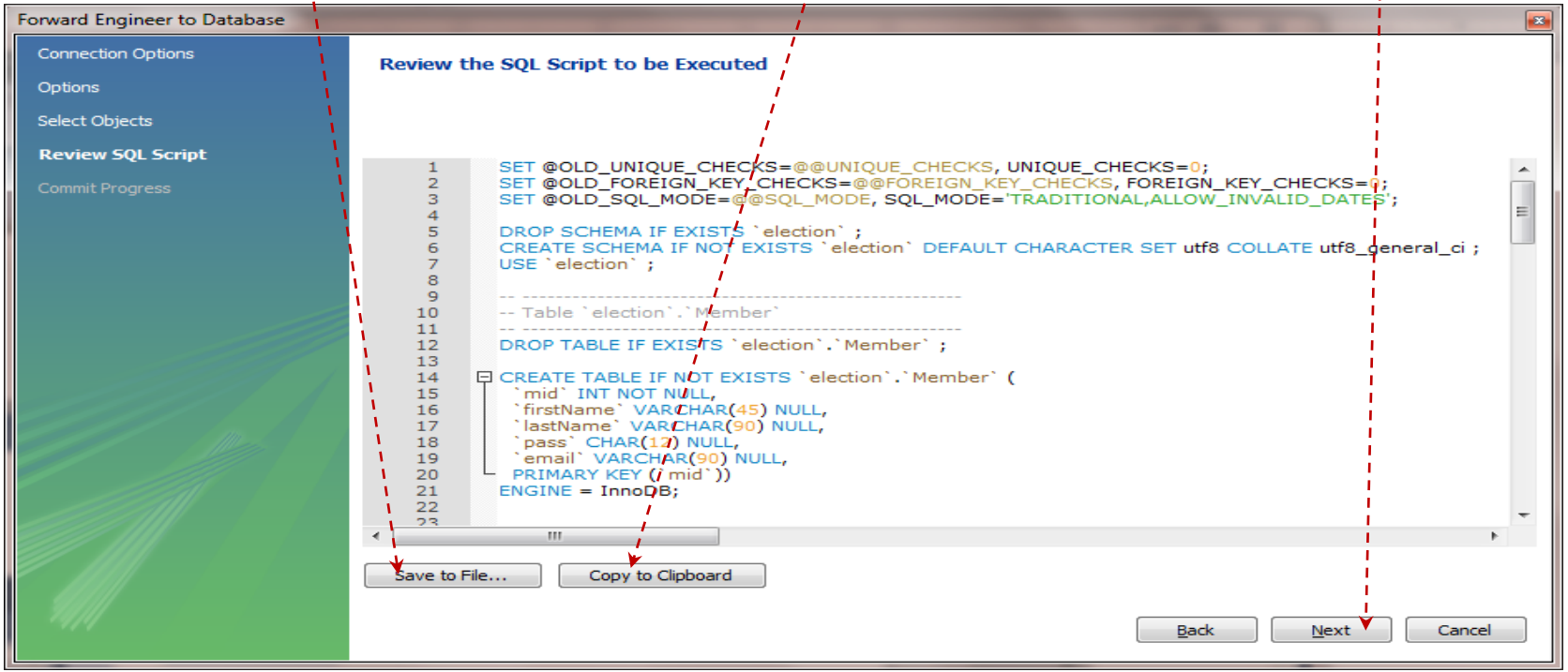


Select (*check*) just one option, **Export MySQL Table Objects**, and then *click* button **Next**.



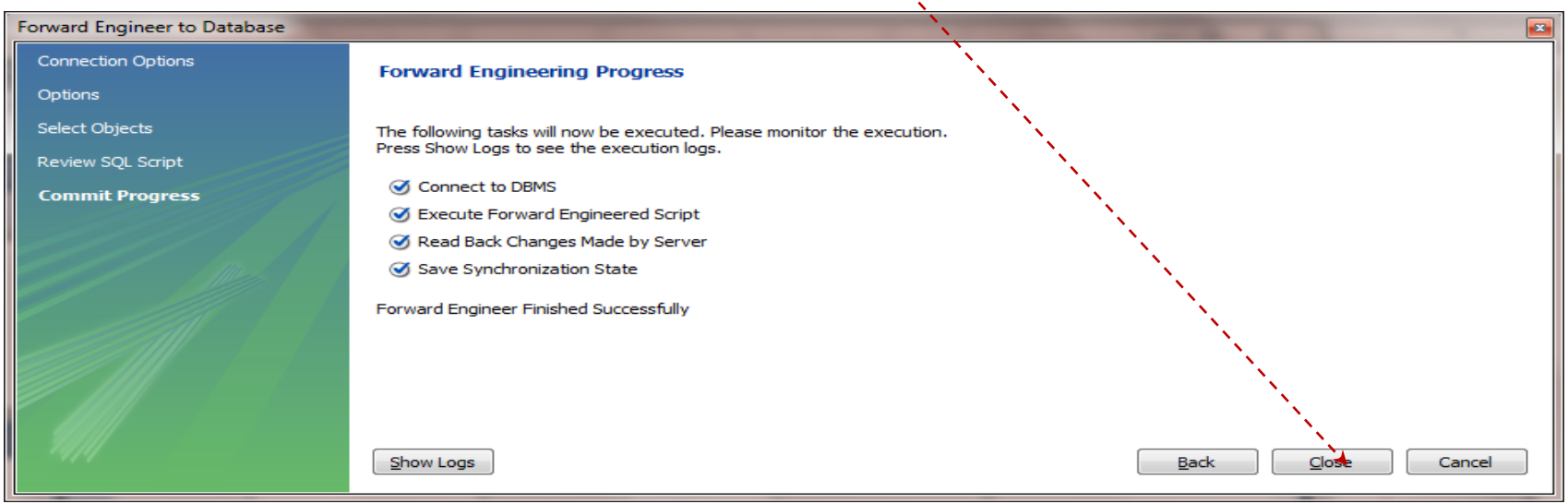
MW generates an **SQL** script, including all statements necessary to create the database, **election**, tables (**Member**, **Office**, **Ballot**, **Candidate** and **Vote**) as well as indexes of the foreign key for tables (**Ballot**, **Candidate** and **Vote**).

Save the script to a **file**, ***click*** button **Copy to Clipboard**, and then ***click*** button **Next**.



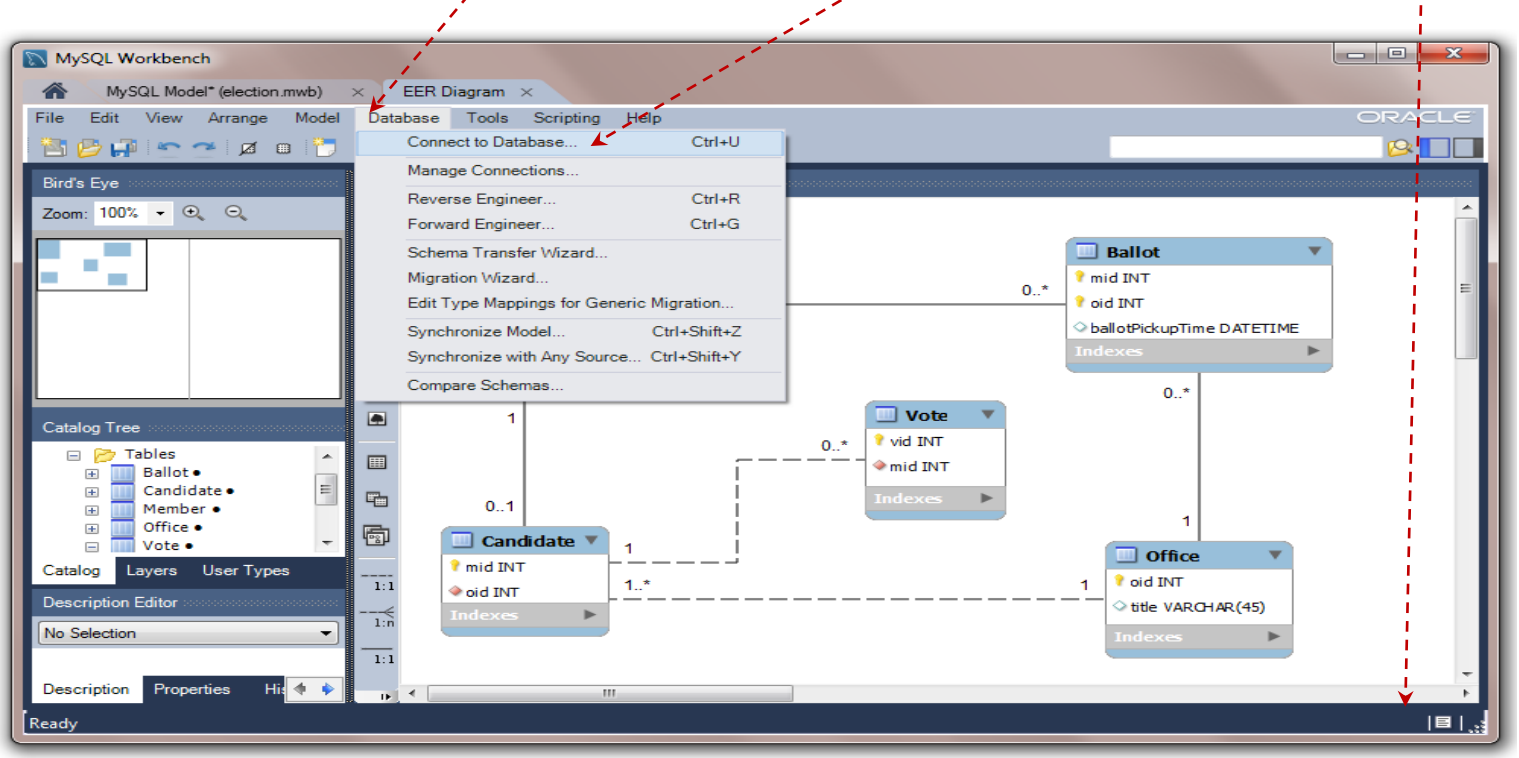
MW connects to the server (**DBMS**) and executes the **SQL** script. The logical design has just been transformed [automatically] into a physical database.

Click button **Close**.

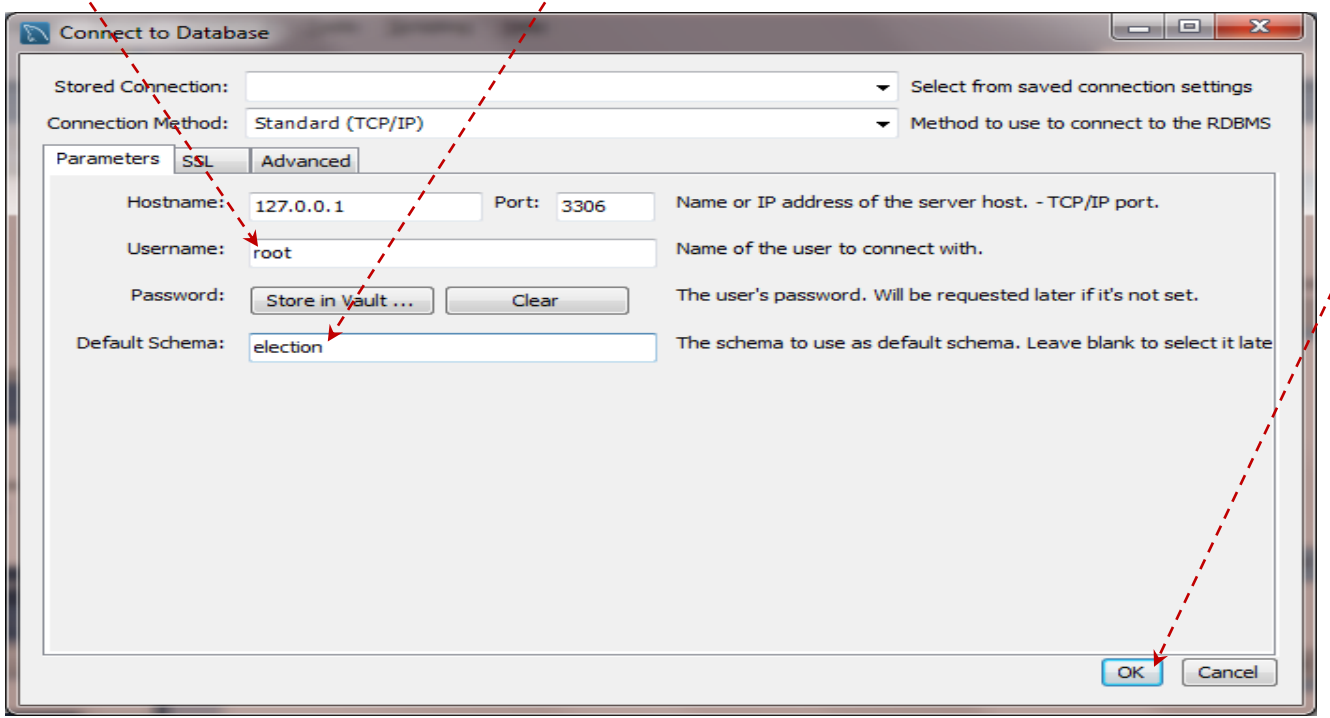


Why don't we connect to the **election** database and add a few records to the tables. The **election.xlsm** workbook contains sample **Insert** statements.

Select (***click***) menu options **Database** and **Connect to Database ...** .

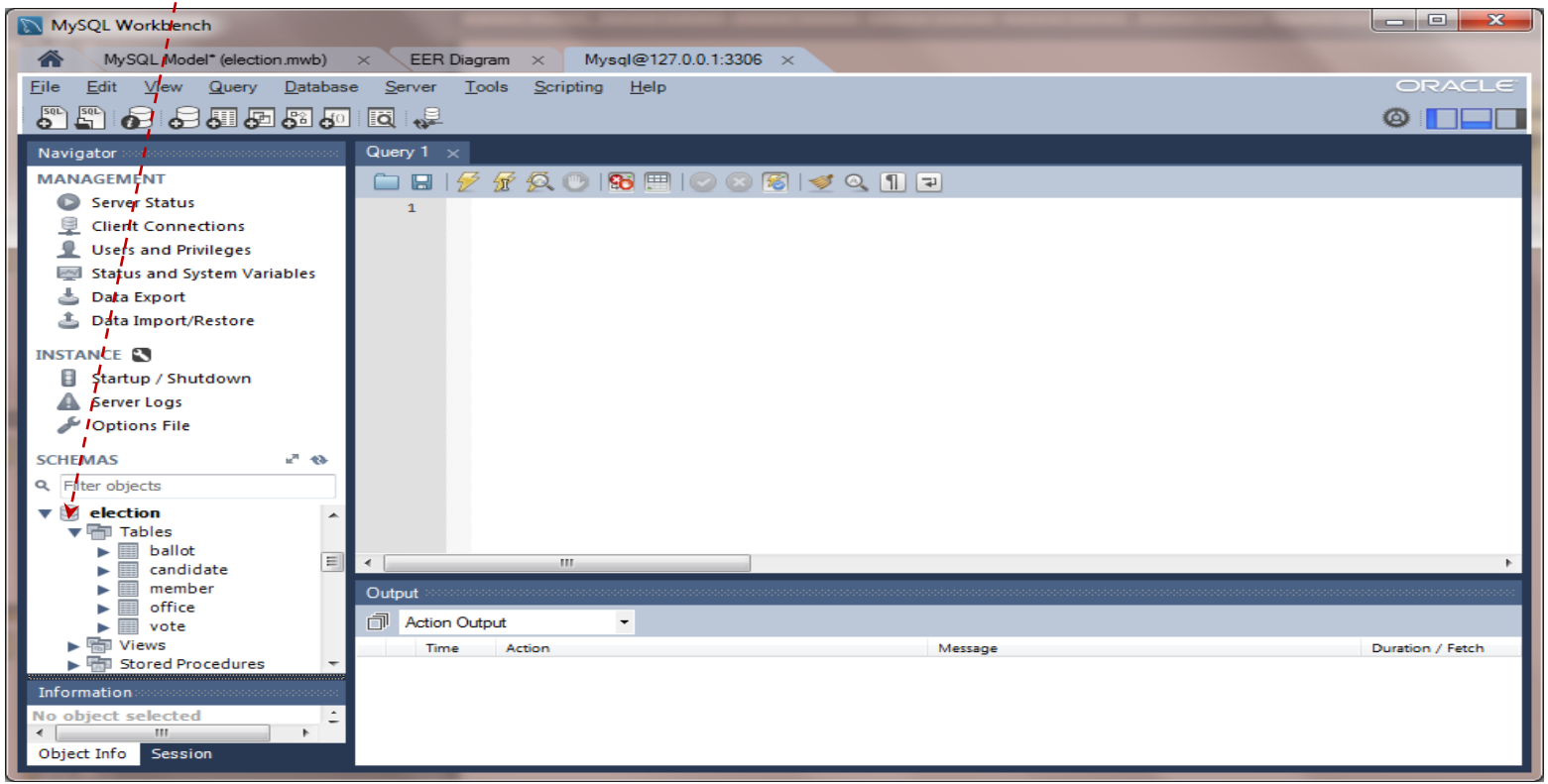


Use the **root** connection. *Type* name **election** for Default Schema and then *click* button **OK**.



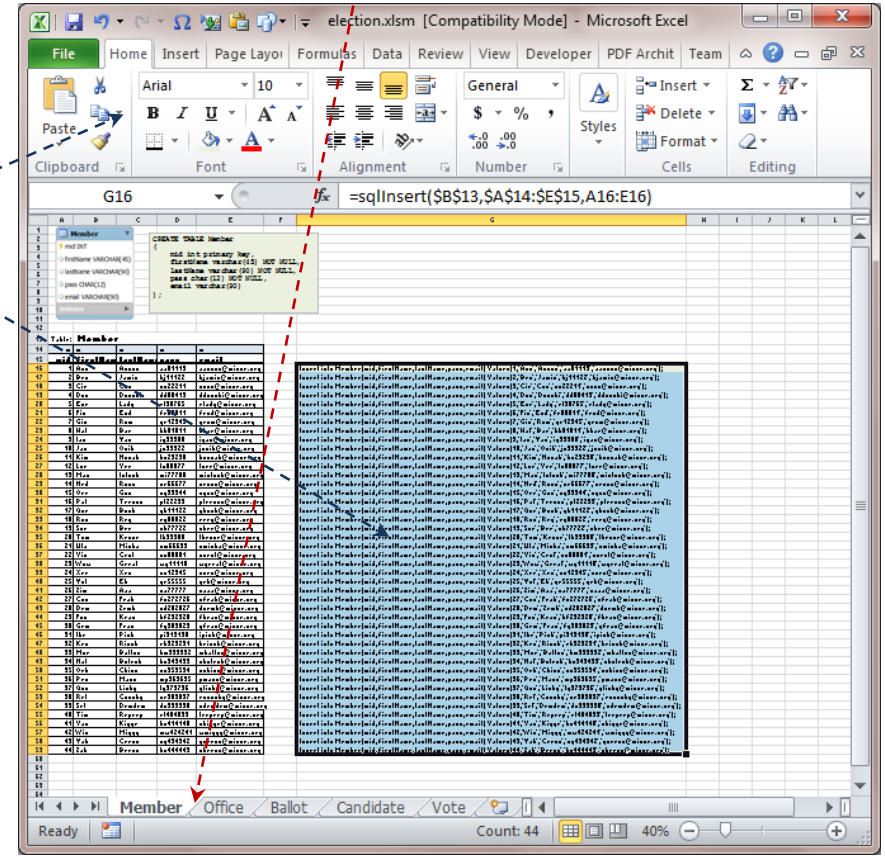
MW opens a Query panel and reveals all existing databases in the **SCHEMAS** panel. A full expansion of the **election** tree will show more details, including column names.

Expand the **election** tree in order to reveal the tables.



Switch to the **election.xlsm** workbook and select the **Member** worksheet.

Select and ***copy*** all the **Insert Statements**.



Switch back to the **Query** panel in **MW** and *press* **Ctrl+V**, in order to paste the **Insert** statements.

MySQL Workbench

MySQL Model* (election.mwb) | EER Diagram | Mysql@127.0.0.1:3306

File Edit View Query Database Server Tools Scripting Help

Navigator

MANAGEMENT
- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE
- Startup / Shutdown
- Server Logs
- Options File

SCHEMAS

Filter objects

- election
  - Tables
    - ballot
      - Columns
        - mid
        - oid
        - ballotPickupTime

Information

No object selected

Object Info | Session

Query 1

```
28  Insert into Member(mid,firstName,lastName,pass,email) Values(28,'Dem','Zemb','zd2828
29  Insert into Member(mid,firstName,lastName,pass,email) Values(29,'Fox','Kran','kf2929
30  Insert into Member(mid,firstName,lastName,pass,email) Values(30,'Gem','Fran','fg3030
31  Insert into Member(mid,firstName,lastName,pass,email) Values(31,'Ike','Pick','pi3131
32  Insert into Member(mid,firstName,lastName,pass,email) Values(32,'Ken','Rinck','rk323
33  Insert into Member(mid,firstName,lastName,pass,email) Values(33,'Mor','Batton','bm33
34  Insert into Member(mid,firstName,lastName,pass,email) Values(34,'Nat','Buteck','bn34
35  Insert into Member(mid,firstName,lastName,pass,email) Values(35,'Orb','Chinu','co353
36  Insert into Member(mid,firstName,lastName,pass,email) Values(36,'Pen','Manu','mp3636
37  Insert into Member(mid,firstName,lastName,pass,email) Values(37,'Qun','Linky','lq373
38  Insert into Member(mid,firstName,lastName,pass,email) Values(38,'Ret','Connky','cr38
39  Insert into Member(mid,firstName,lastName,pass,email) Values(39,'Set','Demdem','ds39
40  Insert into Member(mid,firstName,lastName,pass,email) Values(40,'Tim','Reprep','rt40
41  Insert into Member(mid,firstName,lastName,pass,email) Values(41,'Van','Kigge','kv414
42  Insert into Member(mid,firstName,lastName,pass,email) Values(42,'Win','Miggy','mw424
43  Insert into Member(mid,firstName,lastName,pass,email) Values(43,'Yak','Ceron','cy434
44  Insert into Member(mid,firstName,lastName,pass,email) Values(44,'Zak','Beron','bz444
45
```

Output

Action Output

Time	Action	Message	Duration / Fetch

With all the **Insert** statements in the **Query** panel, *click* the **Execute** button.

The **Output** panel shows feedback for each of the executes statements.

MySQL Workbench

MySQL Model* (election.mwb) | EER Diagram | Mysql@127.0.0.1:3306

File Edit View Query Database Server Tools Scripting Help

Navigator

MANAGEMENT
- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE
- Startup / Shutdown
- Server Logs
- Options File

SCHEMAS

Filter objects

- election
  - Tables
    - ballot
      - Columns
        - mid
        - oid
        - ballotPickupTime

Information

No object selected

Object Info | Session

Query 1

```
28 Insert into Member(mid,firstName,lastName,pass,email) Values(28,'Dem','Zemb','zd2828
29 Insert into Member(mid,firstName,lastName,pass,email) Values(29,'Fox','Kran','kf2929
30 Insert into Member(mid,firstName,lastName,pass,email) Values(30,'Gem','Fran','fg3030
31 Insert into Member(mid,firstName,lastName,pass,email) Values(31,'Ike','Pick','pi3131
32 Insert into Member(mid,firstName,lastName,pass,email) Values(32,'Ken','Rinck','rk323
33 Insert into Member(mid,firstName,lastName,pass,email) Values(33,'Mor','Batton','bm33
34 Insert into Member(mid,firstName,lastName,pass,email) Values(34,'Nat','Buteck','bn34
35 Insert into Member(mid,firstName,lastName,pass,email) Values(35,'Orb','Chinu','co353
36 Insert into Member(mid,firstName,lastName,pass,email) Values(36,'Pen','Manu','mp3636
37 Insert into Member(mid,firstName,lastName,pass,email) Values(37,'Qun','Linky','lq373
38 Insert into Member(mid,firstName,lastName,pass,email) Values(38,'Ret','Connky','cr38
39 Insert into Member(mid,firstName,lastName,pass,email) Values(39,'Set','Demdem','ds39
40 Insert into Member(mid,firstName,lastName,pass,email) Values(40,'Tim','Reprep','rt40
41 Insert into Member(mid,firstName,lastName,pass,email) Values(41,'Van','Kigge','kv414
42 Insert into Member(mid,firstName,lastName,pass,email) Values(42,'Win','Miggy','mw424
43 Insert into Member(mid,firstName,lastName,pass,email) Values(43,'Yak','Ceron','cy434
44 Insert into Member(mid,firstName,lastName,pass,email) Values(44,'Zak','Beron','bz444
45
```

Output

Action Output

	Time	Action	Message	Duration / Fetch
41	11:50:03	Insert into Member(mid,firstName,lastName,pa...	1 row(s) affected	0.000 sec
42	11:50:03	Insert into Member(mid,firstName,lastName,pa...	1 row(s) affected	0.000 sec
43	11:50:03	Insert into Member(mid,firstName,lastName,pa...	1 row(s) affected	0.015 sec
44	11:50:03	Insert into Member(mid,firstName,lastName,pa...	1 row(s) affected	0.000 sec

In order to clear the **Query** panel, ***select*** all the **Insert** statements (***press*** **Ctrl+A**) and ***press*** the **Delete** key.

In a similar way, bring the other **Insert** statements (for tables **Office**, **Ballot**, **Candidate** and **Vote**) from the Excel workbook, **election.xlsm** , to the **MW**'s **Query** panel. Execute the statements and explore the database by running a few queries.

After all the sample records have been added, run a few queries. The first example shows all positions stored in the **Office** table.

Type statement `SELECT * FROM Office;` into the `Query` panel and *click* the `Execute` button.

Query 1

```
1  SELECT * FROM Office;
```

Result Set Filter: | Edit: | Export/Import: | Wrap Cell Content:

oid	title
1	President
2	Vice-President
3	Treasurer
4	Editor
5	Web Manager
NULL	NULL

Office 4 | Apply | Cancel

Output

Action Output

	Time	Action	Message	Duration / Fetch
1	12:23:05	SELECT * FROM Office LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec

Query result!

Query status!

The second example shows all the candidates.

Type statement

```
SELECT * FROM Member WHERE mid IN (SELECT mid FROM Candidate);
```

into the **Query** panel

and *click* the **Execute** button.

Query 1

```
1  SELECT * FROM Member WHERE mid IN (SELECT mid FROM Candidate);
```

Result Set Filter: Edit: Export/Import: Wrap Cell Co

mid	firstName	lastName	pass	email
3	Cir	Cus	cc22211	ccus@misor.org
13	Mac	Intosh	mi77788	mintosh@misor.org
22	Vin	Cent	cv00001	vcent@misor.org
4	Don	Donski	dd00413	ddonski@misor.org
10	Jan	Osik	jo33322	josik@misor.org
15	Ore	Gon	og33344	ogon@misor.org
24	Xer	Xes	xx12345	xxes@misor.org
8	Hal	Bar	hb01011	hbar@misor.org

Member 5 Apply Cancel

Output

Action Output

	Time	Action	Message	Duration / Fetch
1	12:23:05	SELECT * FROM Office LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
2	12:30:00	SELECT * FROM Member WHERE mid IN (SE...	20 row(s) returned	0.000 sec / 0.000 sec

The same result can by done by an explicit JOIN statement:

```
SELECT * FROM Member JOIN Candidate USING(mid);
```

The third example shows all the candidates and offices they are running for.

Type statement

```
SELECT * FROM Member JOIN Candidate USING(mid) JOIN Office USING(oid);
```

into the **Query** panel

and *click* the **Execute** button.

Query 1

```
SELECT * FROM Member JOIN Candidate USING(mid) JOIN Office USING(oid);
```

Result Set Filter: Export: Wrap Cell Content:

oid	mid	firstName	lastName	pass	email	title
1	3	Cir	Cus	cc22211	ccus@misor.org	President
1	13	Mac	Intosh	mi77788	mintosh@misor.org	President
1	22	Vin	Cent	cv00001	vcent@misor.org	President
2	4	Don	Donski	dd00413	ddonski@misor.org	Vice-President
2	10	Jan	Osik	jo33322	josik@misor.org	Vice-President
2	15	Ore	Gon	og33344	ogon@misor.org	Vice-President
2	24	Xer	Xes	xx12345	xxes@misor.org	Vice-President
3	8	Hal	Bar	hb01011	hbar@misor.org	Treasurer

Result 7 Read Only

Output

Action Output

	Time	Action	Message	Duration / Fetch
2	12:30:00	SELECT * FROM Member WHERE mid IN (...	20 row(s) returned	0.000 sec / 0.000 sec
3	12:34:39	SELECT * FROM Member JOIN Candidate U...	20 row(s) returned	0.000 sec / 0.000 sec
4	12:35:54	SELECT * FROM Member JOIN Candidate U...	20 row(s) returned	0.000 sec / 0.000 sec

The fourth example shows all the members who have not yet picked any of their ballots.

Type statement

```
SELECT mid, lastName FROM Member WHERE mid NOT IN (SELECT mid FROM Ballot);
```

into the **Query** panel
and *click* the **Execute** button.

Query 1

```
SELECT mid, lastName FROM Member WHERE mid NOT IN (SELECT mid FROM Ballot);
```

Result Set Filter: Edit: Export/Import: Wrap Cell Content:

mid	lastName
5	Lady
19	Bee
29	Kran
42	Miggy
NULL	NULL

Using an existential query:

```
SELECT mid, lastName
FROM Member
WHERE NOT EXISTS
(SELECT * FROM Ballot WHERE Ballot.mid=Member.mid);
```

Member 9 Apply Cancel

Output

Action Output

	Time	Action	Message	Duration / Fetch
4	12:35:54	SELECT * FROM Member JOIN Candidate U...	20 row(s) returned	0.000 sec / 0.000 sec
5	13:25:40	SELECT mid, lastName FROM Member WH...	5 row(s) returned	0.000 sec / 0.000 sec
6	13:38:36	SELECT mid, lastName FROM Member WH...	4 row(s) returned	0.000 sec / 0.000 sec

This final example shows the number of positions the members have already voted for.

Type statement

```
SELECT mid, Count(Ballot.mid) FROM Ballot GROUP BY mid;
```

into the **Query** panel

and *click* the **Execute** button.

Query 1

```
1  SELECT mid, Count(Ballot.mid) FROM Ballot GROUP BY mid;
2
```

Result Set Filter: Export: Wrap Cell Content:

mid	Count(Ballot.mid)
38	5
39	5
40	5
41	5
43	5
44	5
45	1

Result 10 Read Only

Output

Action Output

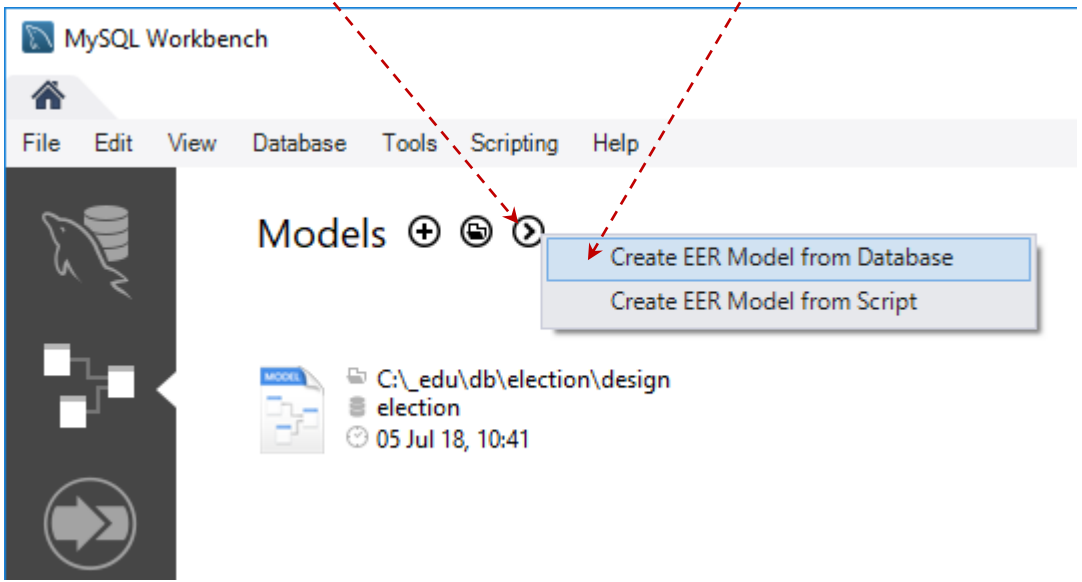
	Time	Action	Message	Duration / Fetch
5	13:25:40	SELECT mid, lastName FROM Member WH...	5 row(s) returned	0.000 sec / 0.000 sec
6	13:38:36	SELECT mid, lastName FROM Member WH...	4 row(s) returned	0.000 sec / 0.000 sec
7	13:43:33	SELECT mid, Count(Ballot.mid) FROM Ballot ...	41 row(s) returned	0.000 sec / 0.000 sec

Notice that one of the members have picked up only one ballot.

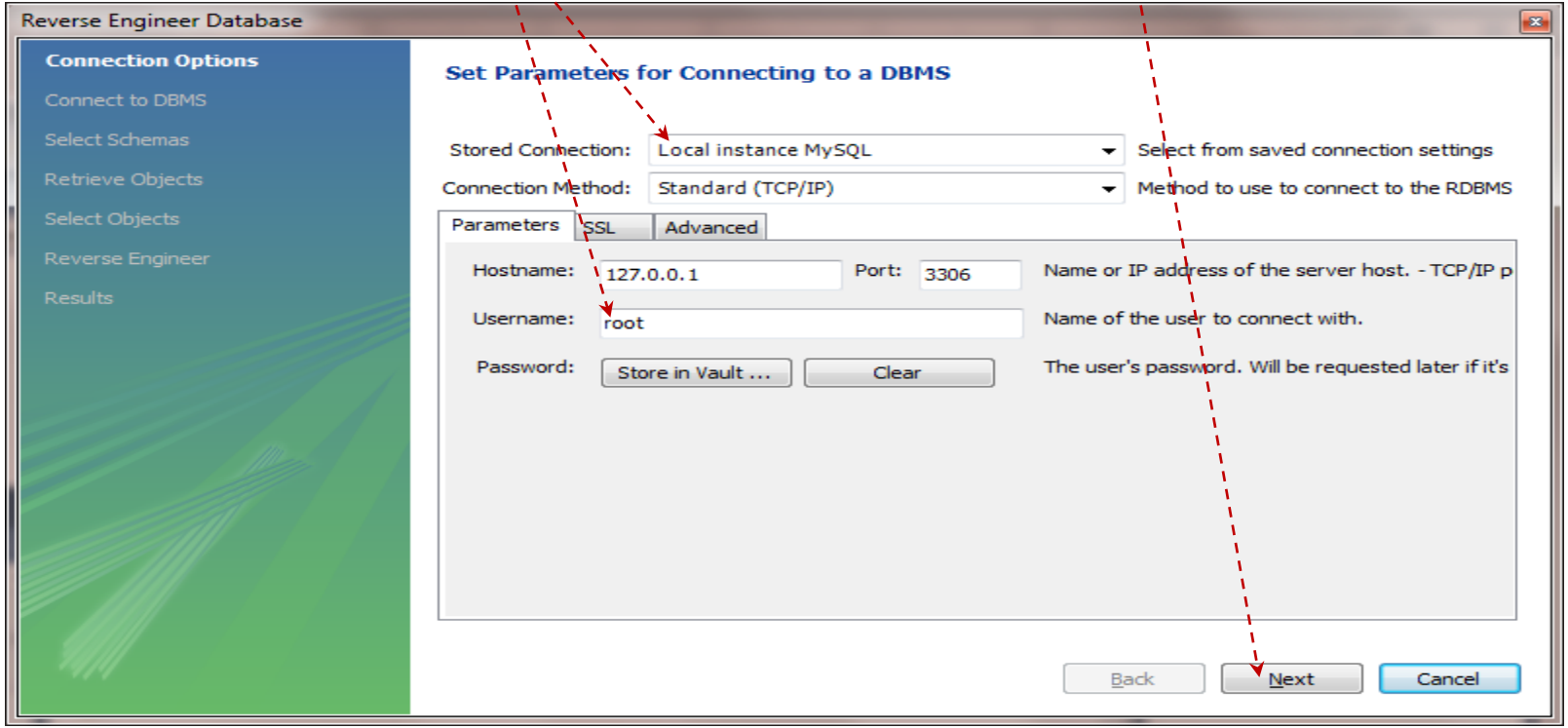
This concludes the **Forward Engineering** case, including the logical design and schema generation, extended by populating the databases with sample records and running queries. The remaining slides show how to **Reverse Engineer** an existing [physical] database. It is a good time to save and close **MW** (**MySQL Workbench**). Then start a new instance of **MW** (a fresh start).

Start **MySQL Workbench** and select the **Data Model** option (see slides 4 and 5 for details).

Click **Button** (>) and select the **Create EER Model from Database**.

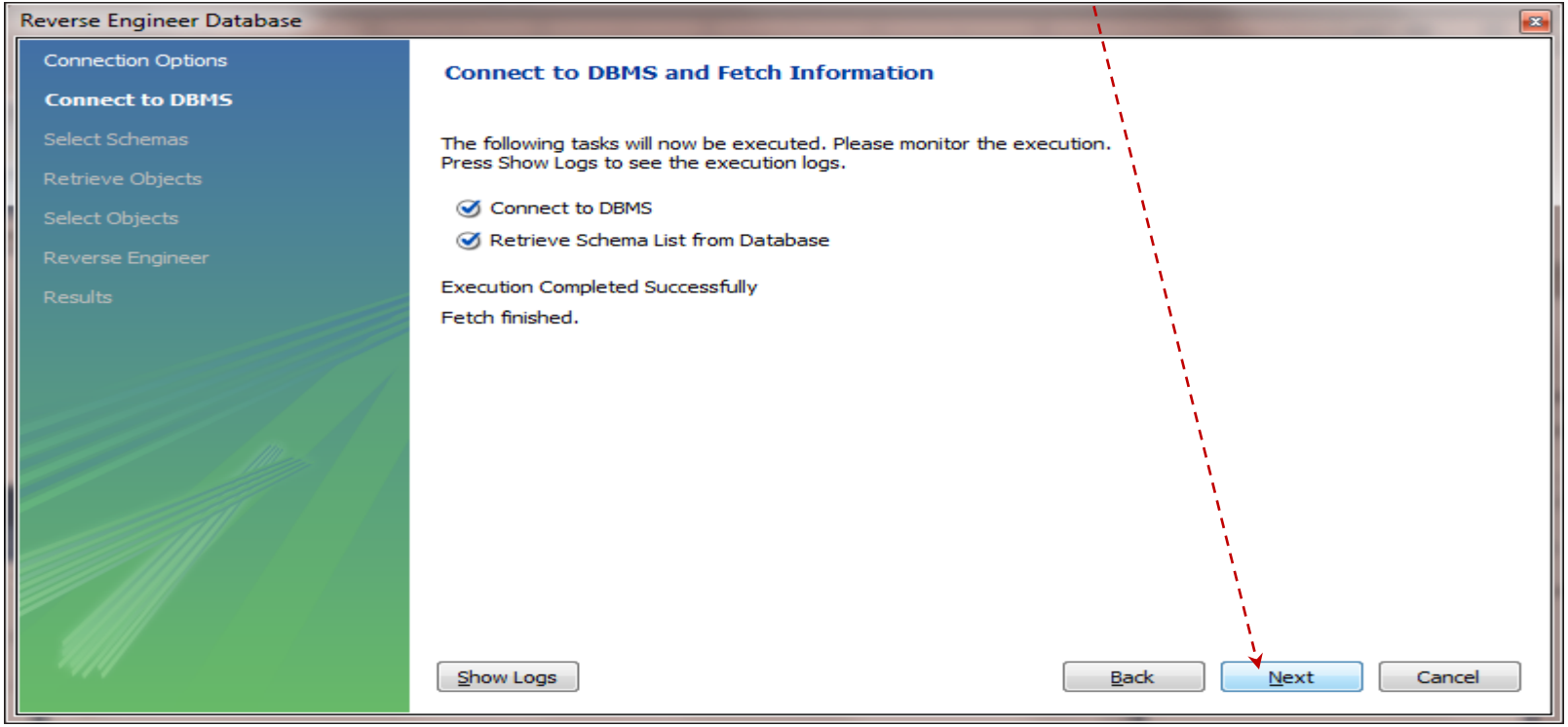


Use your **root** connection and *click* button **Next**.

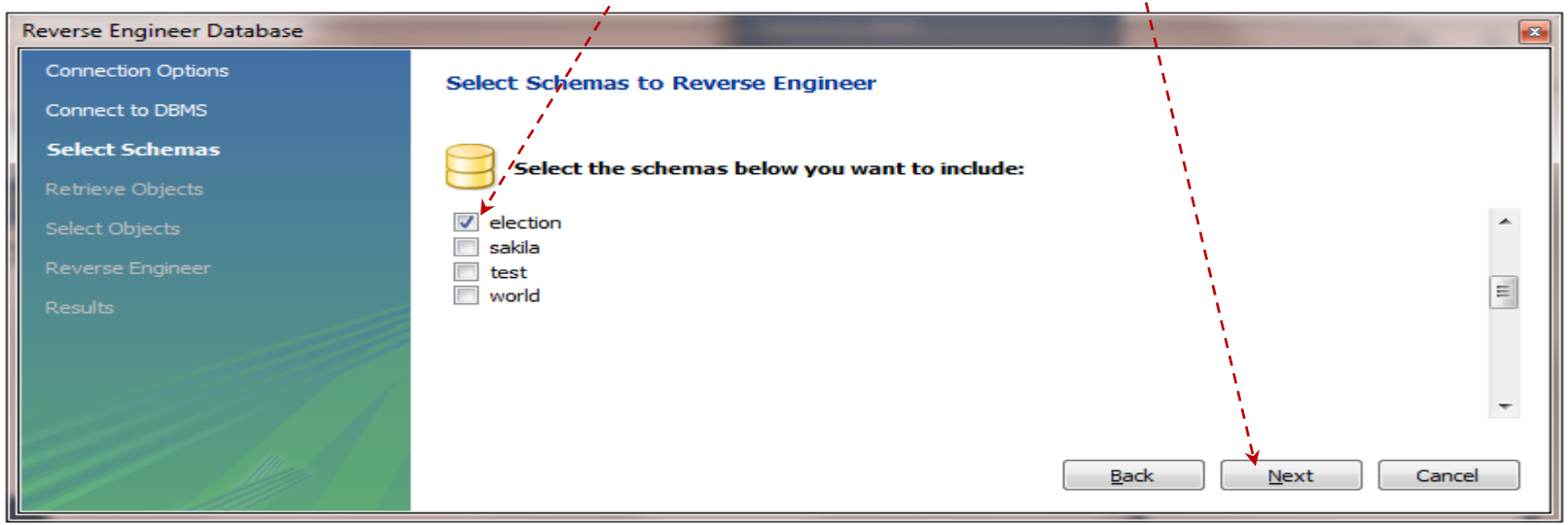


MW is connecting to the server and retrieving the existing schemas from the **DBMS**.

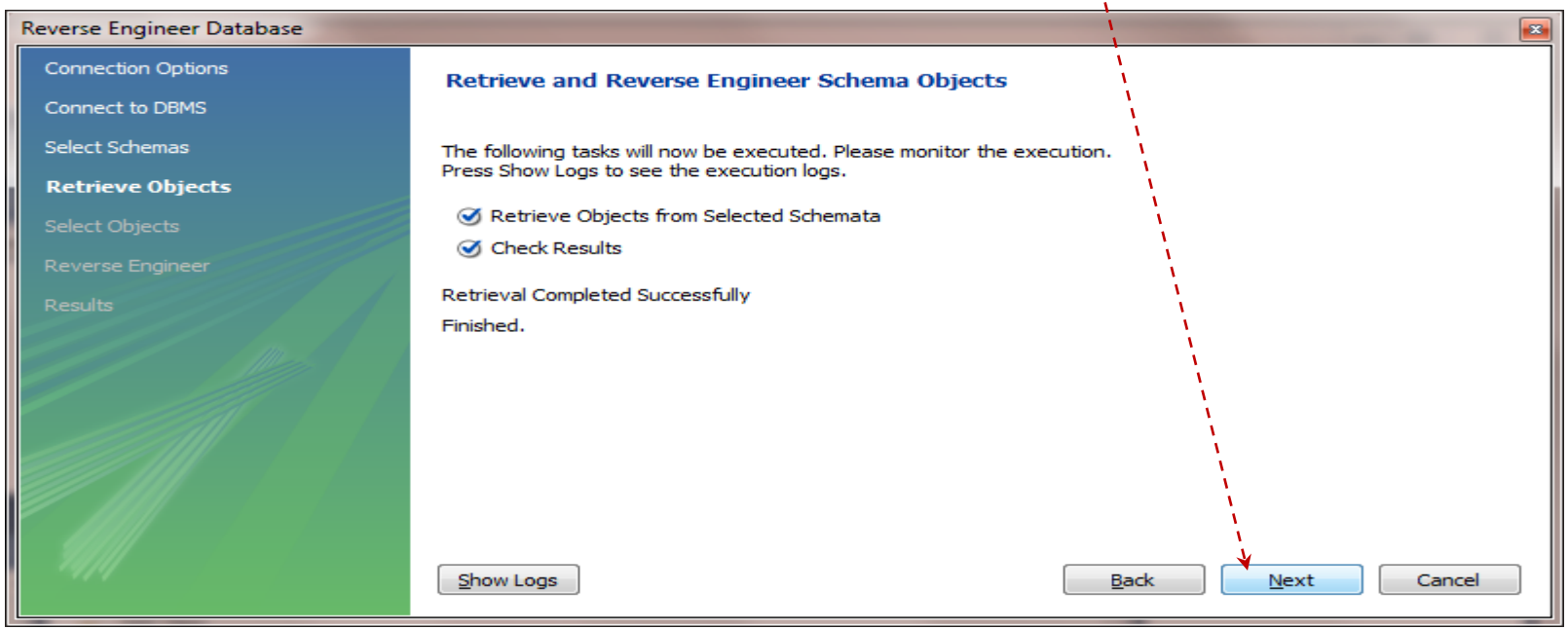
Click button **Next**.



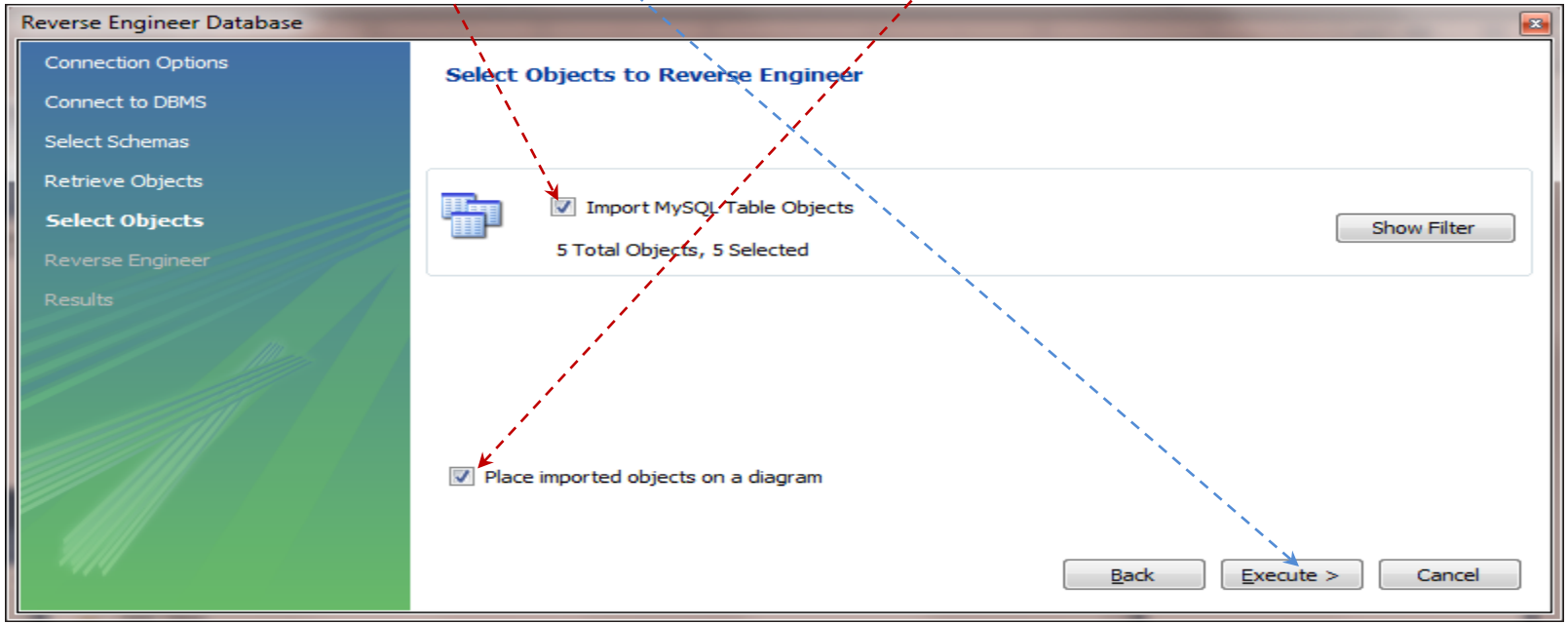
Select database **election** and ***click*** button **Next.**



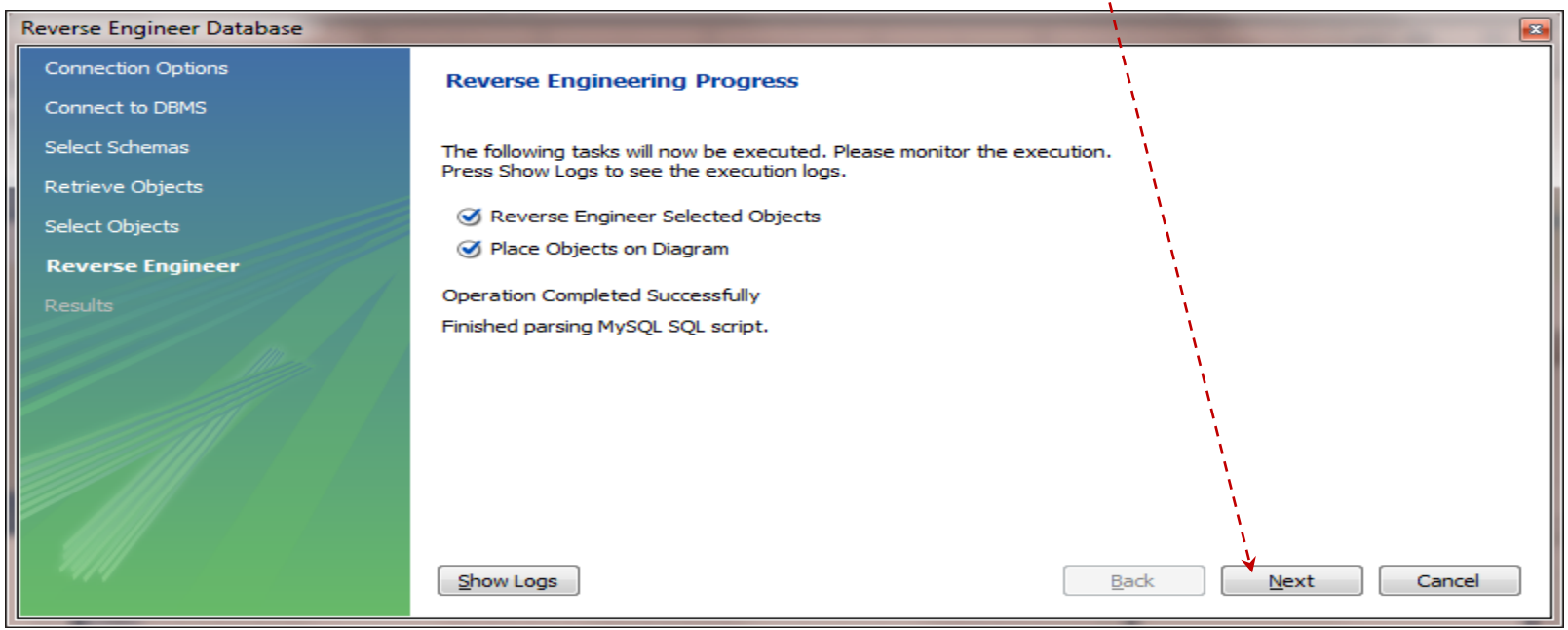
Click button **Next**.



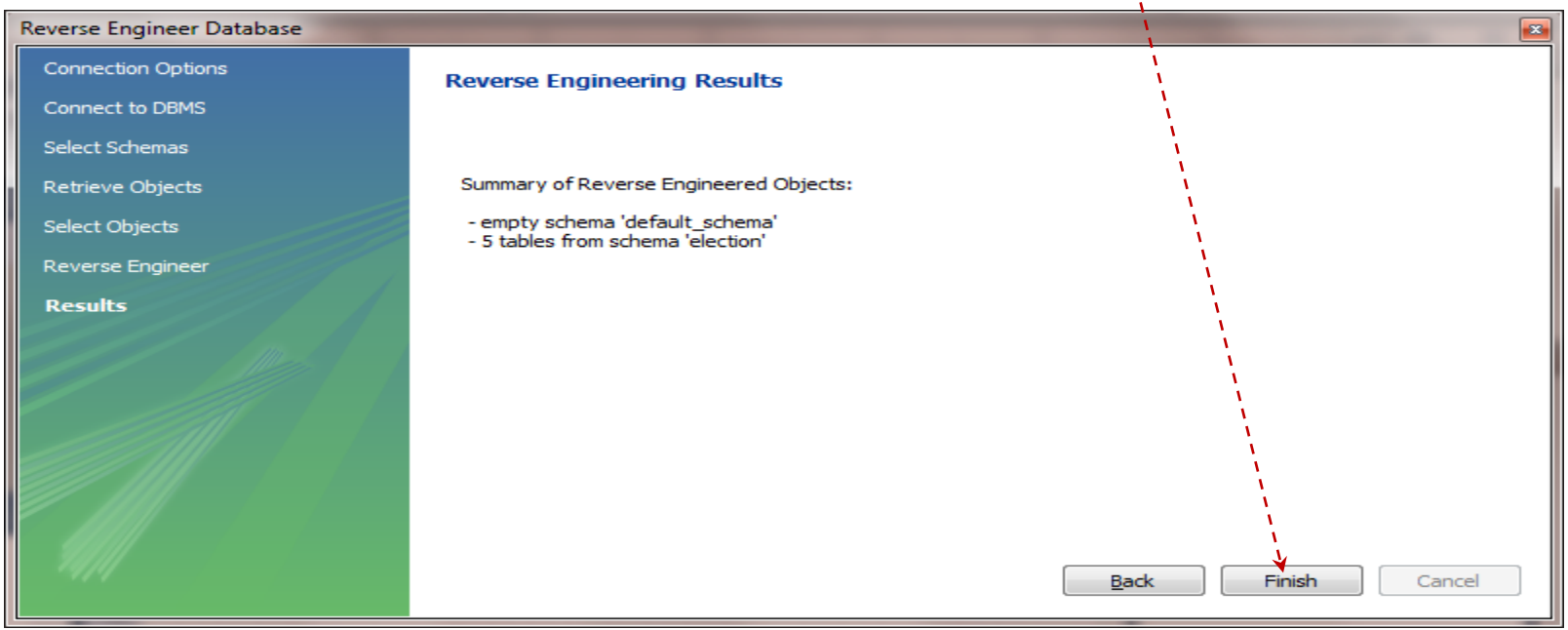
Make sure options **Import MySQL Table Objects** and **Place imported objects on a diagram** are selected and ***click*** button **Execute**.



Click button **Next**.

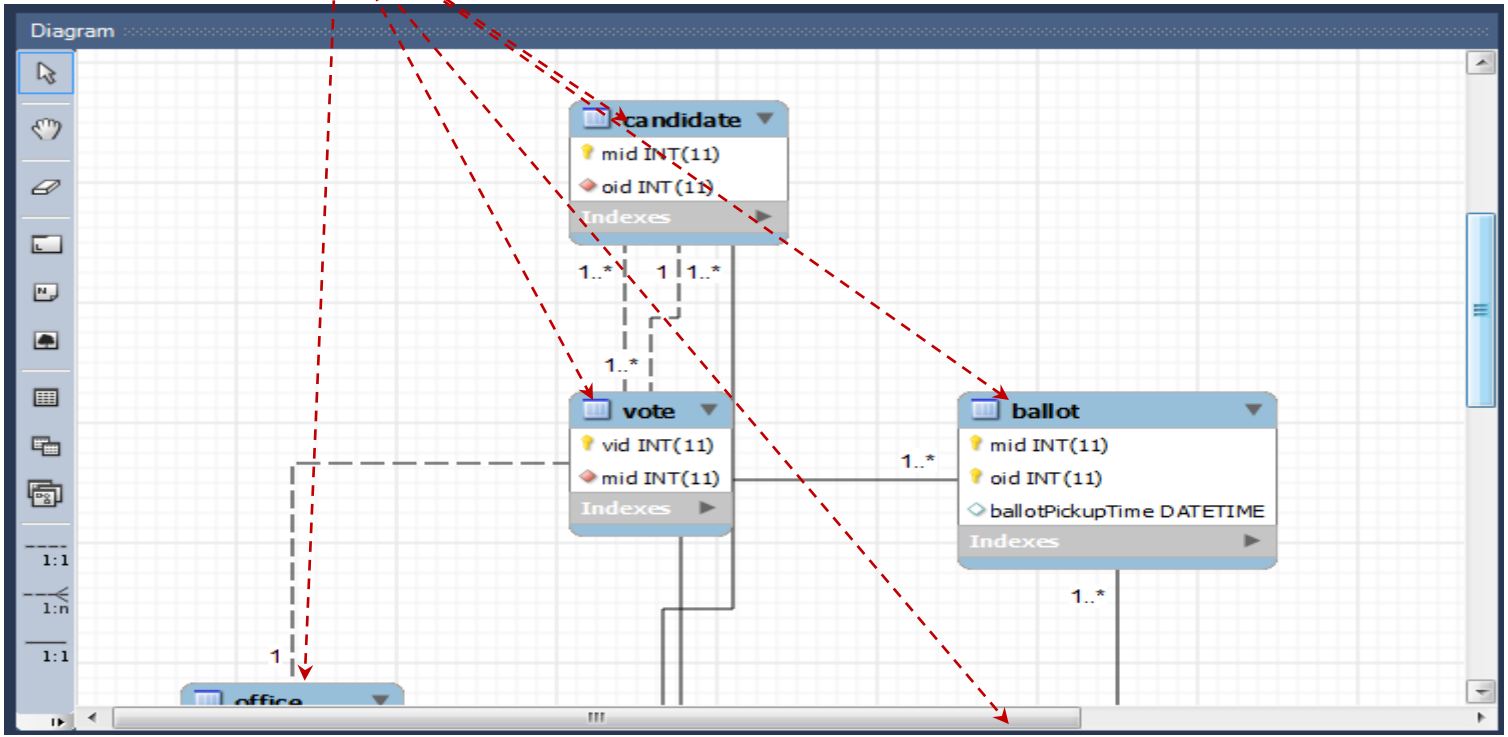


Click button **Finish**.



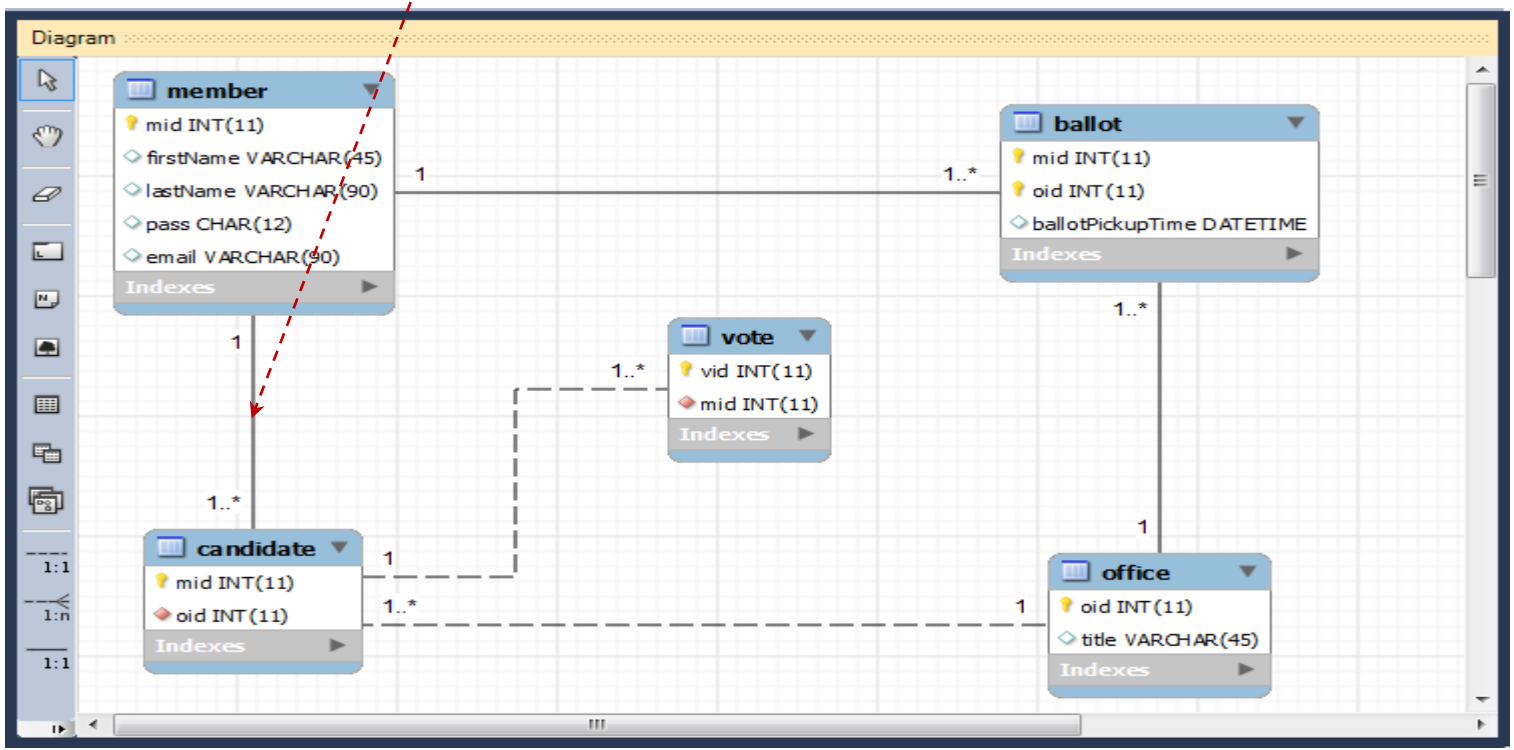
The table objects are not arranged in a friendly way.

Drag the **tables** around in order to better organize the diagram.



Notice that the diagram is almost identical to the original one (developed manually). The critical difference is that the relationship between **Member** and **Candidate** is shown as One-to-Many (**1:1..***).

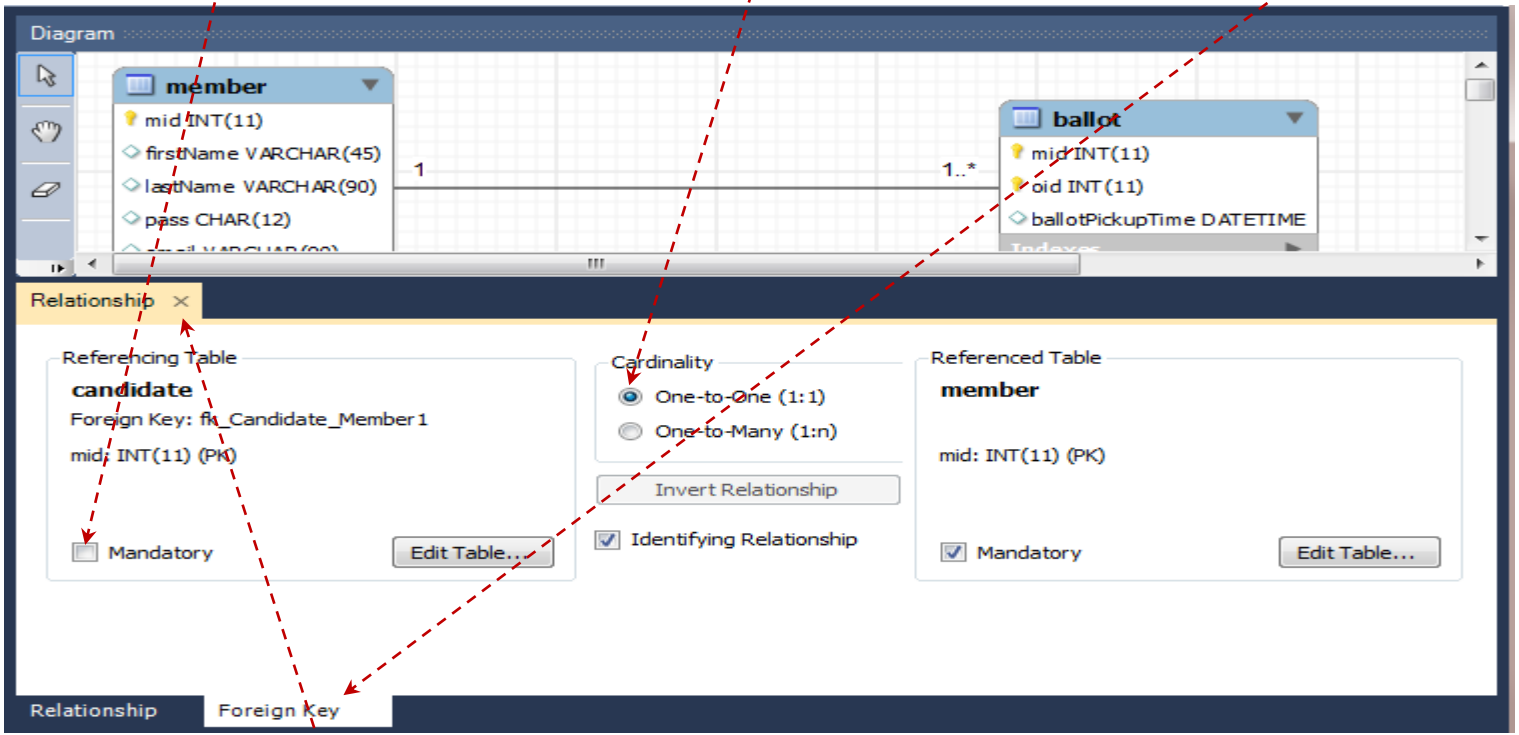
Double-click the **Member - Candidate** relationship link.



Click the **Foreign Key** tab.

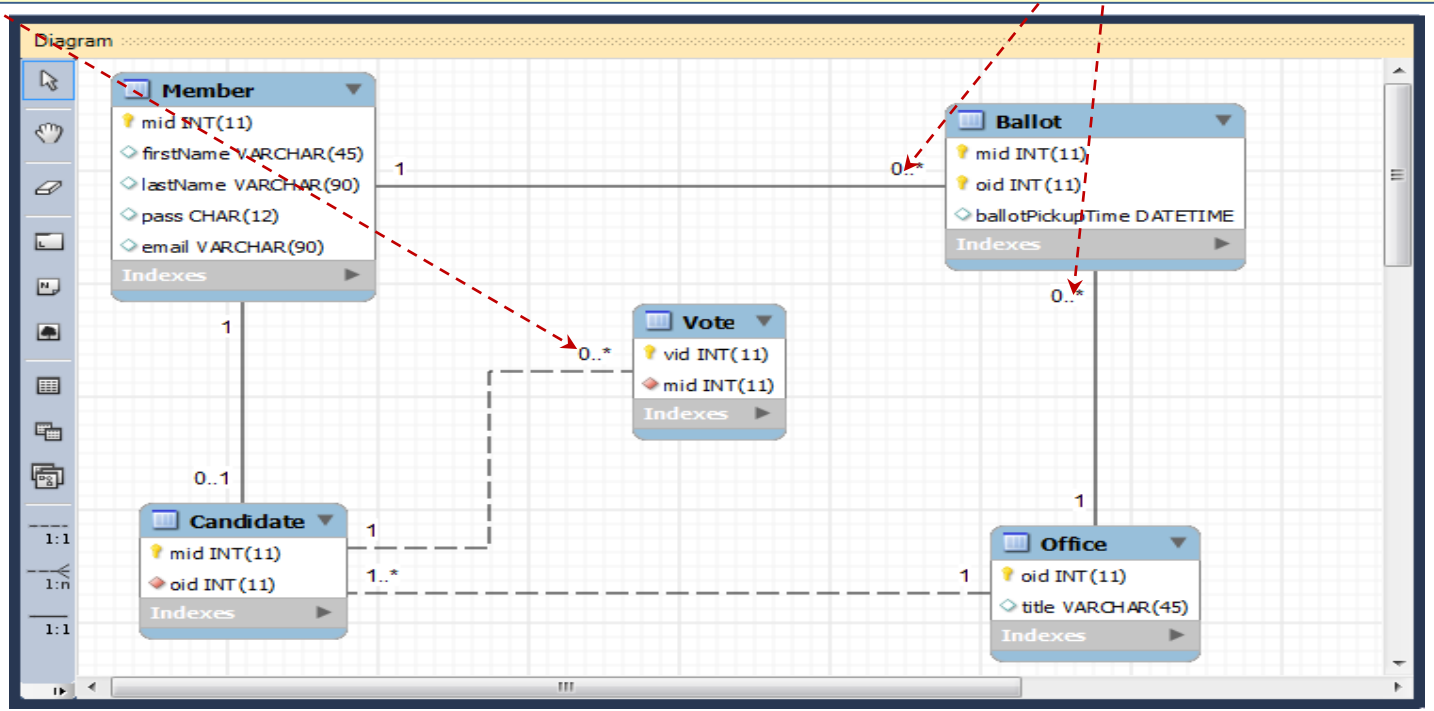
Uncheck the **Mandatory** option in the **Candidate** panel.

Turn on the **One-to-One (1:1)** cardinality option.



When done, *close* the **Relationship** panel.

Using similar operations, modify the **Member – Ballot**, **Ballot – Office** and **Candidate – Vote** relationships. Make sure that the **Mandatory** participation of entities **Ballot** and **Vote** are turned **off** so that they will all show up as **0..*** (Optional-Many).



A perfectionist would also show the table names in the title case. Finally, save the model and close **MW**.

This is it!