

# UML State Machines

## Schema Relazionale

StateMachine(SID, Name, Description, ABSPath, *LastState*, Author)

StateVertex(VID, SID, Name)

State(StateID, VID, Visibility, Description, isFinal, isTop)

PseudoState(*VID*, Type)

CompositeState(*StateID*, isConcurrent, *childSID*)

Transition(TransitionID, Name, Visibility, *Source*, *Target*)

GuardCondition(*TransitionID*, BooleanExpression)

Action(ActionID, *StateID*, *TransitionID*, Name, Visibility, Type, CallType)

Event(EventID, *StateID*, *TransitionID*, Name, Visibility)

CallEvent(EventID, CallType, Return)

TimeEvent(EventID, When)

SignalEvent(EventID, Return)

ChangeEvent(EventID, BooleanExpression)

TPseudo(Name)

TVisibility(Name)

## Dizionario dei Dati

<i>Entita'</i>	<i>Descrizione</i>	<i>Attributi</i>
<b>Model Element</b>	Rappresentazione degli Elementi che fanno parte del 'core' di UML.	<i>None</i>
<b>State Machine</b>	Rappresenta la macchina a stati, una struttura costituita da una serie di stati a piu' livelli.	<b>Name:</b> Nome della macchina a stati <b>Description:</b> Descrizione della macchina <b>ABSPath:</b> Directory assoluta su disco <b>LastState:</b> Ricorda l'ultimo stato <b>Author:</b> Autore proprietario del progetto
<b>State Vertex</b>	Rappresenta la generale designazione di uno stato.	<b>Name:</b> Nome dello stato
<b>State</b>	Uno stato descrive una situazione durante cui avvengono una o piu' azioni.	<b>Visibility:</b> Scoping dell'elemento <b>Description:</b> Descrizione dello stato
<b>Composite State</b>	Rappresenta uno stato costituito da una o piu' sotto-macchine.	<b>isConcurrent:</b> Variabile booleana per il parallelismo
<b>Pseudo State</b>	Rappresenta gli elementi appartenenti alla categoria "Pseudo Stati" di UML.	<b>Type:</b> Descrizione del tipo di pseudo-stato
<b>Transition</b>	Rappresenta una relazione diretta tra due vertici. Porta la macchina da uno stato ad un altro a seguito di un determinato evento.	<b>Name:</b> Nome della transazione <b>Visibility:</b> Scoping dell'elemento
<b>Guard Condition</b>	E' la condizione che viene valutata prima di effettuare una transazione.	<b>BooleanExpression:</b> Espressione di tipo booleano
<b>Action</b>	Rappresentano le attivita' svolte all'ingresso, all'uscita o all'interno di uno stato.	<b>Name:</b> Nome della transazione <b>Visibility:</b> Scoping dell'elemento <b>Type:</b> Tipo di azione <b>CallType:</b> Descrive se di tipo interna o esterna
<b>Event</b>	Quando si verifica, genera la transizione da uno stato ad un altro.	<b>Name:</b> Nome dell'evento <b>Visibility:</b> Scoping dell'elemento

## Bussiness Rules

- Ownership

1. Se una State Machine viene cancellata, verranno cancellati tutti gli states vertex facenti parte della macchina, i relativi stati, le transizioni, azioni ed eventi.
2. Se uno state vertex viene cancellato, verranno cancellate in cascata tutte le transizioni ad esso associate.
3. Se un composite state viene cancellato, verranno cancellate tutte le macchine a stati ad esso associate.

- Naming

1. Il nome di una State Machine deve essere unico e non puo' essere replicato.
2. Stati definiti nella stessa State Machine devono avere necessariamente nomi diversi.

- State Machines

1. L'attributo ABSPath appartenente allo schema delle State Machines deve essere necessariamente non nullo.
2. La coppia costituita da Nome e ABSPath deve essere unica.
3. Una State Machine deve avere uno ed uno solo pseudo stato di tipo "Initial".
4. Una State Machine deve avere al più uno stato di tipo "Final".

- States

1. Se uno stato ha valore logico '1' per l'attributo "isFinal" o l'attributo "isTop", il medesimo non puo' essere uno pseudo stato.
2. Se uno stato ha valore logico '1' per l'attributo "isFinal" o l'attributo "isTop", il medesimo non puo' essere un composite state.
3. Un Composite State non puo' avere come figlio la State Machine a cui appartiene.
4. Uno stato non puo' avere gli attributi "isTop" e "isFinal" contemporaneamente al valore logico '1'.
5. Se uno stato ha valore logico '1' per l'attributo "isTop" non può essere "Source" o "Target" in una transizione.
6. Se uno stato ha valore logico '1' per l'attributo "isFinal" non può avere transizioni in uscita.

- Pseudo States

1. Uno pseudo stato di tipo "History" o "Exit" non può avere transizioni in uscita.
2. Uno pseudo stato di tipo "Initial" o "Entry" non può avere transizioni in ingresso.
3. Uno pseudo stato di tipo "Join" può avere al più una transizione in uscita.
4. Uno pseudo stato di tipo "Fork" può avere al più una transizione in ingresso.

- Transitions

1.  $\forall x \in \text{TRANSITION}, x.\text{source} \neq \text{NIL} \wedge x.\text{target} \neq \text{NIL}$
2. Due transizioni possono avere nomi uguali se differiscono per almeno uno tra Source e Target.
3. Una transizione in uscita da uno pseudo stato di tipo 'Initial' non può avere una guard condition o un evento.

- Events and Actions

1.  $\neg \exists x \in \text{EVENT} : x.\text{StateID} \neq \text{NIL} \wedge x.\text{TransitionID} \neq \text{NIL}$
2.  $\neg \exists x \in \text{ACTION} : x.\text{StateID} \neq \text{NIL} \wedge x.\text{TransitionID} \neq \text{NIL}$

- Properties and User Defined Types

1. L'attributo "Visibility" deve essere scelto dalle entry "Protected, Private, Public, Package"
2. L'attributo "StateType" deve essere scelto dalle entry presenti nella tabella TPpseudo.

## Schema.SQL

```
CREATE TABLE StateMachine(  
  SID integer NOT NULL,  
  Name varchar2(30) NOT NULL,  
  Description varchar2(100),  
  ABSPath varchar2(1000) NOT NULL,  
  Author varchar2(50),  
  CONSTRAINT sm_pk PRIMARY KEY (SID),  
  CONSTRAINT sm_ck1 UNIQUE (Name, ABSPath)  
);  
  
CREATE TABLE StateVertex(  
  VID integer NOT NULL UNIQUE,  
  SID integer NOT NULL,  
  Name varchar2(30),  
  CONSTRAINT ver_pk PRIMARY KEY (VID, SID),  
  CONSTRAINT ver_ck UNIQUE (SID, Name),  
  CONSTRAINT ver_fk1 FOREIGN KEY (SID)  
    REFERENCES StateMachine (SID) ON DELETE CASCADE  
);  
  
CREATE TABLE State(  
  StateID integer PRIMARY KEY,  
  VID integer NOT NULL,  
  Visibility varchar2(15),  
  Description varchar2(100),  
  isFinal integer CHECK (isFinal IN (0, 1)),  
  isTop integer CHECK (isTop IN (0, 1)),  
  CONSTRAINT state_vis CHECK  
    (Visibility IN ('Protected', 'Private', 'Public', 'Package')),  
  CONSTRAINT state_fk FOREIGN KEY (VID)  
    REFERENCES StateVertex(VID) ON DELETE CASCADE  
);  
  
CREATE TABLE PseudoState(  
  VID integer NOT NULL UNIQUE,  
  StateType varchar2(15) NOT NULL,  
  CONSTRAINT ps_fk FOREIGN KEY (VID)  
    REFERENCES StateVertex(VID) ON DELETE CASCADE  
);  
  
CREATE TABLE CompositeState(  
  StateID integer NOT NULL,  
  isConcurrent integer CHECK (isConcurrent IN(0, 1)),
```

```

childSID integer NOT NULL,
CONSTRAINT cs_fk1 FOREIGN KEY (StateID)
    REFERENCES State (StateID) ON DELETE CASCADE,
CONSTRAINT cs_fk3 FOREIGN KEY (childSID)
    REFERENCES StateMachine (SID) ON DELETE CASCADE
);

CREATE TABLE Transition(
    TransitionID integer PRIMARY KEY,
    Name varchar2(30),
    Visibility varchar2(15),
    Source integer NOT NULL,
    Target integer NOT NULL,
    CONSTRAINT tr_vis CHECK
        (Visibility IN ('Protected', 'Private', 'Public', 'Package')),
    CONSTRAINT tr_fk1 FOREIGN KEY (Source)
        REFERENCES StateVertex (VID) ON DELETE CASCADE,
    CONSTRAINT tr_fk2 FOREIGN KEY (Target)
        REFERENCES StateVertex (VID) ON DELETE CASCADE,
    CONSTRAINT tr_ck1 UNIQUE (Name, Source, Target)
);

CREATE TABLE GuardCondition(
    TransitionID integer,
    BooleanExpression varchar2(250),
    CONSTRAINT gc_fk1 FOREIGN KEY (TransitionID)
        REFERENCES Transition (TransitionID) ON DELETE CASCADE
);

CREATE TABLE Action(
    ActionID integer PRIMARY KEY,
    StateID integer,
    TransitionID integer,
    Name varchar2(30),
    Visibility varchar2(15),
    CONSTRAINT ac_vis CHECK
        (Visibility IN ('Protected', 'Private', 'Public', 'Package')),
    CONSTRAINT ac_fk1 FOREIGN KEY (StateID)
        REFERENCES State (StateID) ON DELETE CASCADE,
    CONSTRAINT ac_fk2 FOREIGN KEY (TransitionID)
        REFERENCES Transition (TransitionID) ON DELETE CASCADE
);

CREATE TABLE Event(
    EventID integer PRIMARY KEY,
    StateID integer,
    TransitionID integer,
    Name varchar2(30),
    Visibility varchar2(15),
    CONSTRAINT ev_vis CHECK
        (Visibility IN ('Protected', 'Private', 'Public', 'Package')),

```

```
CONSTRAINT ev_fk1 FOREIGN KEY (StateID)
    REFERENCES State (StateID) ON DELETE CASCADE,
CONSTRAINT ev_fk2 FOREIGN KEY (TransitionID)
    REFERENCES Transition (TransitionID) ON DELETE CASCADE
);
```

```
CREATE TABLE CallEvent(
    EventID integer NOT NULL,
    CallType varchar2(50),
    Return varchar2(50),
    CONSTRAINT cev_fk1 FOREIGN KEY (EventID)
        REFERENCES Event (EventID) ON DELETE CASCADE
);
```

```
CREATE TABLE TimeEvent(
    EventID integer NOT NULL,
    When varchar2(50),
    CONSTRAINT tev_fk1 FOREIGN KEY (EventID)
        REFERENCES Event (EventID) ON DELETE CASCADE
);
```

```
CREATE TABLE SignalEvent(
    EventID integer NOT NULL,
    Return varchar2(50),
    CONSTRAINT sev_fk1 FOREIGN KEY (EventID)
        REFERENCES Event (EventID) ON DELETE CASCADE
);
```

```
CREATE TABLE ChangeEvent(
    EventID integer NOT NULL,
    BooleanExpression varchar2(50),
    CONSTRAINT chev_fk1 FOREIGN KEY (EventID)
        REFERENCES Event (EventID) ON DELETE CASCADE
);
```

```
CREATE TABLE TPpseudo(
    Name varchar2(50)
);
```

```
INSERT ALL
    INTO TPpseudo VALUES('Initial')
    INTO TPpseudo VALUES('Terminate')
    INTO TPpseudo VALUES('Junction')
    INTO TPpseudo VALUES('Choice')
    INTO TPpseudo VALUES('Join')
    INTO TPpseudo VALUES('Fork')
    INTO TPpseudo VALUES('History')
    INTO TPpseudo VALUES('Entry')
    INTO TPpseudo VALUES('Exit')
SELECT * FROM DUAL;
```

# Triggers.SQL

- Una State Machine deve avere uno ed uno solo pseudo stato di tipo “Initial”.

```
CREATE OR REPLACE TRIGGER ps_initial
BEFORE INSERT OR UPDATE ON PseudoState
FOR EACH ROW
WHEN (new.StateType = 'Initial')
DECLARE
    x INTEGER;
BEGIN
    SELECT COUNT(*) INTO x
    FROM StateVertex S NATURAL JOIN PseudoState P
    WHERE S.SID = (SELECT SID FROM StateVertex S1 WHERE S1.VID = :new.VID);
    IF (x >= 1) THEN
        raise_application_error(-20001, 'Found multiple Initial Vertex. ');
    END IF;
END;
/
```

- Una State Machine deve avere al più uno stato di tipo “Final”.

```
CREATE OR REPLACE TRIGGER state_final
BEFORE INSERT OR UPDATE ON State
FOR EACH ROW
WHEN (new.isFinal = '1')
DECLARE
    x INTEGER;
    actual_sid INTEGER;
BEGIN
    SELECT SID INTO actual_sid
    FROM StateMachine M NATURAL JOIN StateVertex V
    WHERE new.VID = V.VID;

    SELECT COUNT(*) INTO x
    FROM State S NATURAL JOIN StateVertex V
    WHERE actual_sid = V.SID AND S.isFinal = '1';

    IF (x >= 1) THEN
        raise_application_error(-20002, 'Found multiple Final states for given SID. ');
    END IF;
END;
/
```

- Se uno stato ha valore logico ‘1’ per l’attributo “isFinal” o l’attributo “isTop”, il medesimo non può essere uno pseudo stato.

```
CREATE OR REPLACE TRIGGER ps_FinalTop
BEFORE INSERT OR UPDATE ON PseudoState
```

```

FOR EACH ROW
DECLARE
  x INTEGER;
BEGIN
  SELECT COUNT(*) INTO x
  FROM State S
  WHERE :new.StateID = S.StateID AND (S.isFinal = 1 OR S.isTop = 1);
  IF (x>=1) THEN
    raise_application_error(-20003, 'Given VID is either Final or Top.');
```

- Se uno stato ha valore logico '1' per l'attributo "isFinal" o l'attributo "isTop", il medesimo non può essere un composite state.

```

CREATE OR REPLACE TRIGGER cs_FinalTop
BEFORE INSERT OR UPDATE ON CompositeState
FOR EACH ROW
DECLARE
  x INTEGER;
BEGIN
  SELECT COUNT(*) INTO x
  FROM State S
  WHERE :new.StateID = S.StateID AND (S.isFinal = 1 OR S.isTop = 1);
  IF (x>=1) THEN
    raise_application_error(-20004, 'Given VID is either Final or Top.');
```

- Uno stato non può avere gli attributi "isTop" e "isFinal" contemporaneamente al valore logico '1'.

```

CREATE OR REPLACE TRIGGER state_contemp
BEFORE INSERT OR UPDATE ON State
FOR EACH ROW
BEGIN
  IF (:new.isTop = 1 AND :new.isFinal = 1) THEN
    raise_application_error(-20005, 'Given VID is either Final and Top.');
```

- Un Composite State non può avere come figlio la State Machine a cui appartiene.

```

CREATE OR REPLACE TRIGGER cs_inception
BEFORE INSERT OR UPDATE ON CompositeState
```



```

FOR EACH ROW
DECLARE
    actual_sid INTEGER;
BEGIN
    SELECT V.SID INTO actual_sid
    FROM State S NATURAL JOIN StateVertex V
        WHERE S.StateID = :new.StateID;
    IF(actual_sid = :new.childSID) THEN
        raise_application_error(-20006, 'Child SID cannot be equal to VID SID.');
```

- Se uno stato ha valore logico '1' per l'attributo "isTop" non può essere "Source" o "Target" in una transizione.

```

CREATE OR REPLACE TRIGGER tr_Top
BEFORE INSERT OR UPDATE ON Transition
FOR EACH ROW
DECLARE
    x INTEGER;
BEGIN
    SELECT Count(*) INTO x
    FROM State S
        WHERE (:new.Source = S.VID OR :new.target = S.VID) AND S.isTop = 1;
    IF(x>=1) THEN
        raise_application_error(-20007, 'Given Source or Target is a Top State.');
```

- Se uno stato ha valore logico '1' per l'attributo "isFinal" non può avere transizioni in uscita.

```

CREATE OR REPLACE TRIGGER tr_Final
BEFORE INSERT OR UPDATE ON Transition
FOR EACH ROW
DECLARE
    x INTEGER;
BEGIN
    SELECT Count(*) INTO x
    FROM State S
        WHERE :new.Source = S.VID AND S.isFinal = 1;
    IF(x>=1) THEN
        raise_application_error(-20008, 'Given Source is a Final State.');
```

- Uno pseudo stato di tipo “History” o “Exit” non può avere transizioni in uscita.

```
CREATE OR REPLACE TRIGGER tr_History_Exit
BEFORE INSERT OR UPDATE ON Transition
FOR EACH ROW
DECLARE
    x INTEGER;
BEGIN
    SELECT Count(*) INTO x
    FROM PseudoState P
    WHERE :new.Source = P.VID AND (P.StateType = 'History' OR P.StateType = 'Exit');
    IF(x>=1) THEN
        raise_application_error(-20010, 'Given Source is either an History Pseudo State or an Exit Pseudo State.');
```

- Uno pseudo stato di tipo “Initial” o “Entry” non può avere transizioni in ingresso.

```
CREATE OR REPLACE TRIGGER tr_Initial_Entry
BEFORE INSERT OR UPDATE ON Transition
FOR EACH ROW
DECLARE
    x INTEGER;
BEGIN
    SELECT Count(*) INTO x
    FROM PseudoState P
    WHERE :new.Target = P.VID AND (P.StateType = 'Initial' OR P.StateType = 'Entry');
    IF(x>=1) THEN
        raise_application_error(-20011, 'Given Source is either an Initial Pseudo State or an Entry Pseudo State.');
```

- Uno pseudo stato di tipo “Join” può avere al più una transizione in uscita.

```
CREATE OR REPLACE TRIGGER tr_Join
BEFORE INSERT OR UPDATE ON Transition
FOR EACH ROW
DECLARE
    x INTEGER;
BEGIN
    SELECT Count(*) INTO x
    FROM PseudoState P
    WHERE :new.Source = P.VID AND P.StateType = 'Join';
    IF(x>=1) THEN
        raise_application_error(-20012, 'Outgoing transition already exists for given join pseudo state.');
```

```

END IF;
END;
/

```

- Uno pseudo stato di tipo “Fork” può avere al più una transizione in ingresso.

```

CREATE OR REPLACE TRIGGER tr_Fork
BEFORE INSERT OR UPDATE ON Transition
FOR EACH ROW
DECLARE
  x INTEGER;
BEGIN
  SELECT Count(*) INTO x
    FROM PseudoState P
   WHERE :new.Target = P.VID AND P.StateType = 'Fork';
  IF(x>=1) THEN
    raise_application_error(-20013, 'Incoming transition already exists for given fork pseudo state.');
```

- Una transizione in uscita da uno pseudo stato di tipo ‘Initial’ non può avere una guard condition o un evento.

```

CREATE OR REPLACE TRIGGER ev_Initial
BEFORE INSERT OR UPDATE ON Event
FOR EACH ROW
DECLARE
  x INTEGER;
BEGIN
  SELECT Count(*) INTO x
    FROM (State S NATURAL JOIN PseudoState P) JOIN Transition T ON (T.Source = VID)
   WHERE :new.StateID = S.StateID AND P.StateType = 'Initial';
  IF(x>=1) THEN
    raise_application_error(-20014, 'Outgoing transition from Initial Pseudo State can't have an event.');
```

- Un evento deve avere solo uno dei due attributi “StateID” o “TransitionID” referenziato.

```

CREATE OR REPLACE TRIGGER ev_Check
BEFORE INSERT OR UPDATE ON Event
FOR EACH ROW
BEGIN
  IF (:new.StateID <> NULL AND :new.TransitionID <> NULL) THEN
    raise_application_error(-20015, 'Given event has both StateID and TransitionID.');
```

```

ELSIF (:new.StateID = NULL AND :new.TransitionID = NULL) THEN
    raise_application_error(-20016, 'Given event has both StateID and TransitionID to NULL.');
```

- Un'azione deve avere solo uno dei due attributi "StateID" o "TransitionID" referenziato.

```

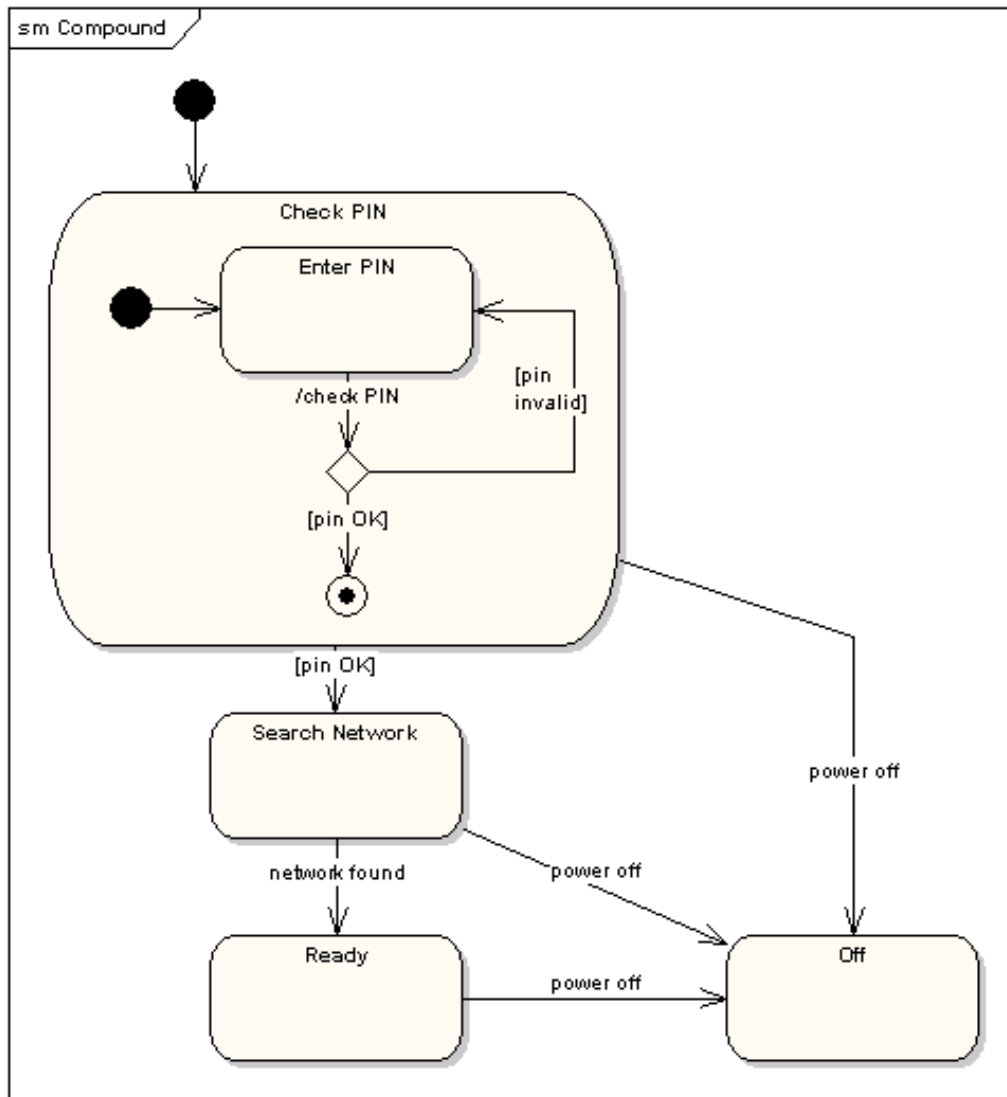
CREATE OR REPLACE TRIGGER ac_Check
BEFORE INSERT OR UPDATE ON Action
FOR EACH ROW
BEGIN
    IF (:new.StateID <> NULL AND :new.TransitionID <> NULL) THEN
        raise_application_error(-20018, 'Given event has both StateID and TransitionID.');
```

- L'attributo "StateType" deve essere scelto dalle entry presenti nella tabella TPpseudo.

```

CREATE OR REPLACE TRIGGER ps_Check
BEFORE INSERT OR UPDATE ON PseudoState
FOR EACH ROW
DECLARE
    x INTEGER;
```

# TestExample.SQL



-- Create StateMachine

```
INSERT INTO StateMachine VALUES(1, 'smCompound', NULL, '/home/user/smcompound', 'Vincenzo Prignano, Valerio Russo');
```

-- Create Top State and Initial

```
INSERT INTO StateVertex VALUES(1, 1, 'smCompoundTop');
INSERT INTO State VALUES(1, 1, 'Package', NULL, 0, 1);
INSERT INTO StateVertex VALUES(7, 1, 'smCompound Initial');
INSERT INTO PseudoState VALUES(7, 'Initial');
```

-- Create EnterPin Composite State

```
INSERT INTO StateVertex VALUES(2, 1, 'Check PIN');
INSERT INTO State VALUES(2, 2, 'Package', NULL, 0, 0);
```

```

INSERT INTO StateMachine VALUES(2, 'smCheckPIN', NULL, '/home/user/smcompound', 'Vincenzo Prignano, Valerio Russo');
INSERT INTO CompositeState VALUES(2, 1, 2);
INSERT INTO StateVertex VALUES(3, 2, 'Check PIN Initial');
INSERT INTO PseudoState VALUES(3, 'Initial');
INSERT INTO StateVertex VALUES(4, 2, 'Enter PIN');
INSERT INTO State VALUES(3, 4, 'Package', NULL, 0, 0);
INSERT INTO StateVertex VALUES(5, 2, 'Enter PIN Choice');
INSERT INTO PseudoState VALUES(5, 'Choice');
INSERT INTO StateVertex VALUES(6, 2, 'Enter PIN Final');
INSERT INTO State VALUES(4, 6, 'Package', NULL, 1, 0);

```

*-- Other states*

```

INSERT INTO StateVertex VALUES(8, 1, 'Search Network');
INSERT INTO State VALUES(5, 8, 'Package', NULL, 0, 0);
INSERT INTO StateVertex VALUES(9, 1, 'Ready');
INSERT INTO State VALUES(6, 9, 'Package', NULL, 0, 0);
INSERT INTO StateVertex VALUES(10, 1, 'Off');
INSERT INTO State VALUES(7, 10, 'Package', NULL, 0, 0);

```

*-- Create transitions*

```

INSERT ALL
  INTO Transition VALUES(1, NULL, 'Package', 7, 2)
  INTO Transition VALUES(2, NULL, 'Package', 3, 4)
  INTO Transition VALUES(3, NULL, 'Package', 4, 5)
  INTO Transition VALUES(4, NULL, 'Package', 5, 4)
  INTO Transition VALUES(5, NULL, 'Package', 5, 6)
  INTO Transition VALUES(6, NULL, 'Package', 2, 8)
  INTO Transition VALUES(7, 'Network Found', 'Package', 8, 9)
  INTO Transition VALUES(8, 'Power Off', 'Package', 8, 10)
  INTO Transition VALUES(9, 'Power Off', 'Package', 9, 10)
  INTO Transition VALUES(10, 'Power Off', 'Package', 2, 10)
SELECT * FROM dual;

```






*-- Insert Guard Conditions, Actions and Events*

```







INSERT ALL
  INTO GuardCondition VALUES(4, 'PIN Invalid')
  INTO GuardCondition VALUES(5, 'PIN OK')
  INTO GuardCondition VALUES(6, 'PIN OK')
  INTO Action VALUES(1, NULL, 3, 'Check PIN', 'Package')
  INTO Event VALUES(1, NULL, 7, 'Network Found', 'Package')
  INTO Event VALUES(2, NULL, 8, 'Power Off', 'Package')
  INTO Event VALUES(3, NULL, 9, 'Power Off', 'Package')
  INTO Event VALUES(4, NULL, 10, 'Power Off', 'Package')
SELECT * FROM dual;

```

StateMachine Table

		SID		NAME		DESCRIPTION		ABSPATH		AUTHOR
1		1		smCompound		(null)		/home/user/smcompound		Vincenzo Prignano, Valerio Russo
2		2		smCheckPIN		(null)		/home/user/smcompound		Vincenzo Prignano, Valerio Russo






State Table

		STATEID		VID		VISIBILITY		DESCRIPTION		ISFINAL		ISTOP
1		1		1		Package		(null)		0		1
2		2		2		Package		(null)		0		0
3		3		4		Package		(null)		0		0
4		5		8		Package		(null)		0		0
5		6		9		Package		(null)		0		0
6		7		10		Package		(null)		0		0
7		4		6		Package		(null)		1		0

CompositeState Table

		STATEID		ISCONCURRENT		CHILDSID
1		2		1		2

Transition Table

		TRANSITIONID		NAME		VISIBILITY		SOURCE		TARGET
1		1		(null)		Package		7		2
2		2		(null)		Package		3		4
3		3		(null)		Package		4		5
4		4		(null)		Package		5		4
5		5		(null)		Package		5		6
6		6		(null)		Package		2		8
7		7		Network Found		Package		8		9
8		8		Power Off		Package		8		10
9		9		Power Off		Package		9		10
10		10		Power Off		Package		2		10

Action Table

	ACTIONID	STATEID	TRANSITIONID	NAME	VISIBILITY
1	1	(null)	3	Check PIN	Package

GuardCondition Table

	TRANSITIONID	BOOLEANEXPRESSION
1	4	PIN Invalid
2	5	PIN OK
3	6	PIN OK

Event Table

	EVENTID	STATEID	TRANSITIONID	NAME	VISIBILITY
1	1	(null)	7	Network Found	Package
2	2	(null)	8	Power Off	Package
3	3	(null)	9	Power Off	Package
4	4	(null)	10	Power Off	Package