From ERD to Relational Model

The acronyms used in this Lab Session: PK – Primary Key, FK – Foreign key, ET – Entity Type.

Basic Conversion Rules

The basic rules convert everything on the ERD except **generalization hierarchies**. You should apply these rules until everything on an ERD is converted except for generalization hierarchies. ***You should use the first 2 rules before the other rules. As you apply these rules, you can use a check mark to indicate the converted parts of an ERD.***

1. **Entity Type Rule:** Each entity type (except subtypes) becomes a table. The PK of ET (if not weak) becomes the PK of the table. The attributes of the ET become columns in the table. This rule **should be used first before the relationship rules**.

2. **1-M Relationship Rule:** Each 1-M relationship becomes a FK in the table corresponding to the child type (the entity type near Crow's Foot symbol). If the minimum cardinality on the parent side of the relationship is one, the FK cannot accept null values ***(NOT NULL must be used).***

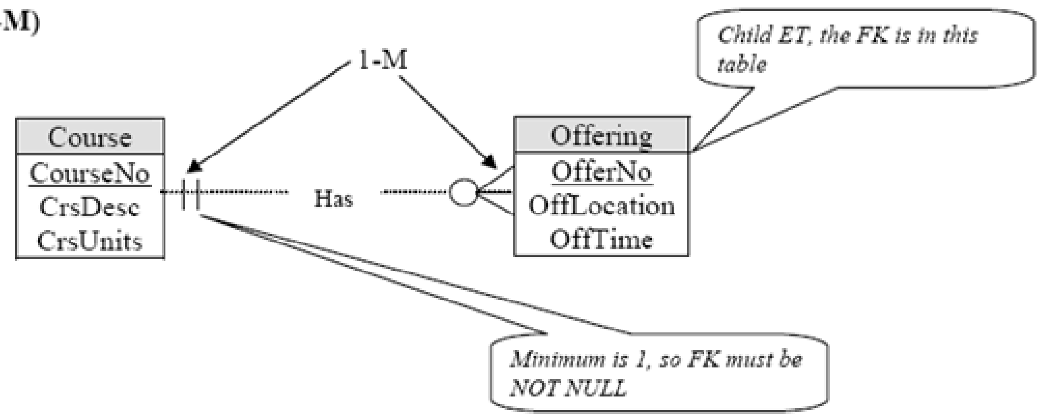
3. **M-N Relationship Rule:** Each M-N relationship becomes a separate table. The **PK of the table is a combined key** consisting of the primary keys of the entity types participating in the M-N relationship.

4. **Identification Dependency Rule:** Each identifying relationship (denoted by a solid relationship line) adds a component to a PK. The PK of the table corresponding to the ***weak entity*** consists of:

    a. The underlined local key (if any) in the weak entity and

    b. The PK(s) of the entity type(s) connected by identifying relationship(s).

EXAMPLE (1-M)



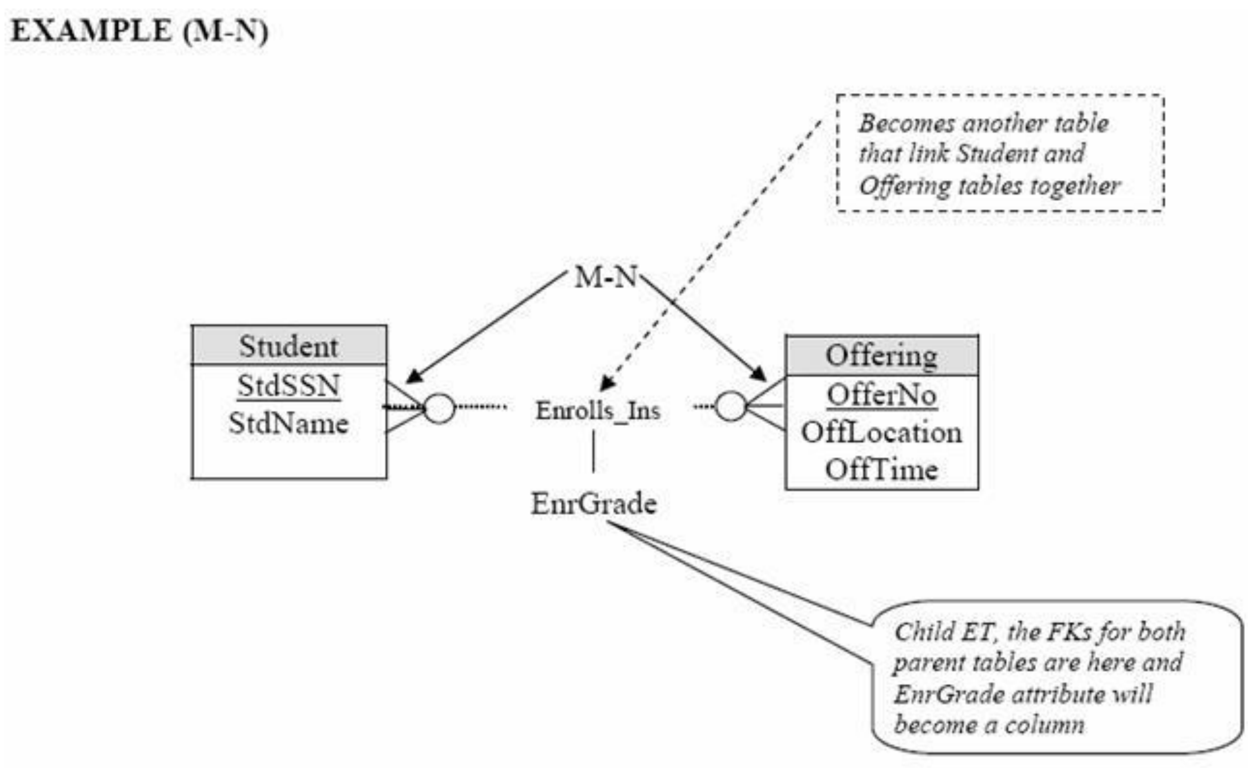

```
CREATE TABLE Course(
      CourseNo      CHAR(6),
      CrsDesc       VARCHAR(30),
      CrsUnits      SMALLINT,
PRIMARY KEY (CourseNo)
);

CREATE TABLE Offering(
      OfferNo      INTEGER,
      OffLocation  CHAR(20),
      CourseNo     CHAR(6) NOT NULL,
      OffTime      TIMESTAMP,
...
PRIMARY KEY (OfferNo),
FOREIGN KEY (CourseNo) REFERENCES Course (CourseNo)
);
```

Converting the generic SQL script to tables we have the following:

Course			
CourseNo (PK)	CrsDesc	CrsUnits	...
...	...	...	...

Offering				
OfferNo (PK)	OffLocation	**CourseNo** (FK)	OffTime	...
...	...	NOT NULL	...	...

Rule 1 is applied to convert the Course and Offering ETs to tables. Then, Rule 2 is applied to convert the Has relationship to a FK (Offering.CourseNo). The Offering table contains the FK because the **Offering ET is the child ET in the Has relationship**. The minimum cardinality on the parent side of the relationship is one, the FK cannot accept null values (**that is why the NOT NULL being used**).

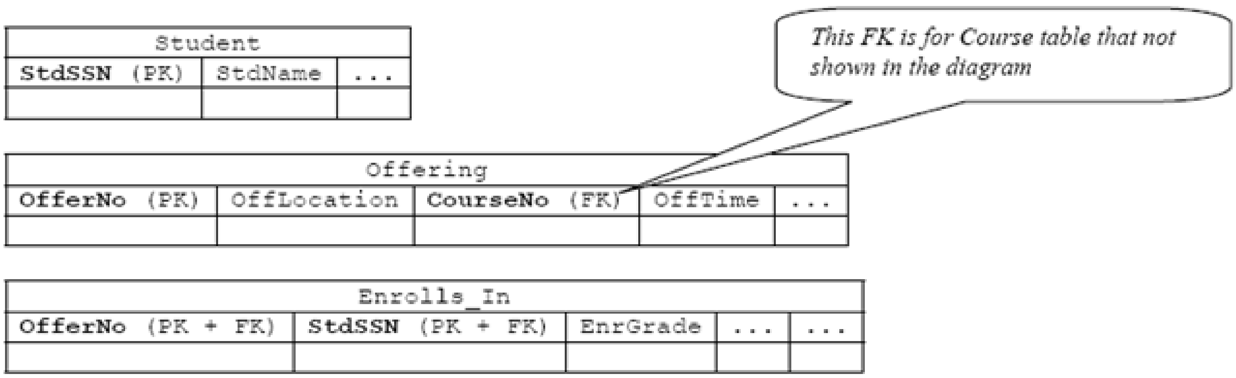


Using Rule 3 leads to the extra Enrolls_in table. The PK of Enrolls_In is a combination of the PKs of the Student and Offering ETs.

```
CREATE TABLE Student(
            StdSSN          CHAR(11),
            StdName         VARCHAR(30),
            ...
            PRIMARY KEY (StdSSN)
);

CREATE TABLE Offering(
            OfferNo         INTEGER,
            OffLocation     VARCHAR(30),
            OffTime         TIMESTAMP,
            ...
            PRIMARY KEY (OfferNo)
);
```

```
CREATE TABLE Enrolls_In(
            OfferNo         INTEGER,
            StdSSN          CHAR(11),
            EnrGrade        DECIMAL(2,1),
            PRIMARY KEY (OfferNo, StdSSN),
            FOREIGN KEY (OfferNo) REFERENCES Offering (OfferNo),
            FOREIGN KEY (StdSSN) REFERENCES Student (StdSSN)
);
```

Converting the generic SQL script to tables we have the following:



For Rule 4, the identification dependency can be used to convert the ERD in the following figure. The result of converting the previous figure is identical to the following figure except that the **Enrolls_In table is renamed Enrollment**. The following figure requires 2 applications of the identification dependency rule. Each application of the identification dependency rule adds a component to the PK of the Enrollment table. So M-N becomes two 1-M relationships.

EXAMPLE (converting the previous ERD diagram with weak ET)

In the following Figure, both sides will become 1-M relationship.

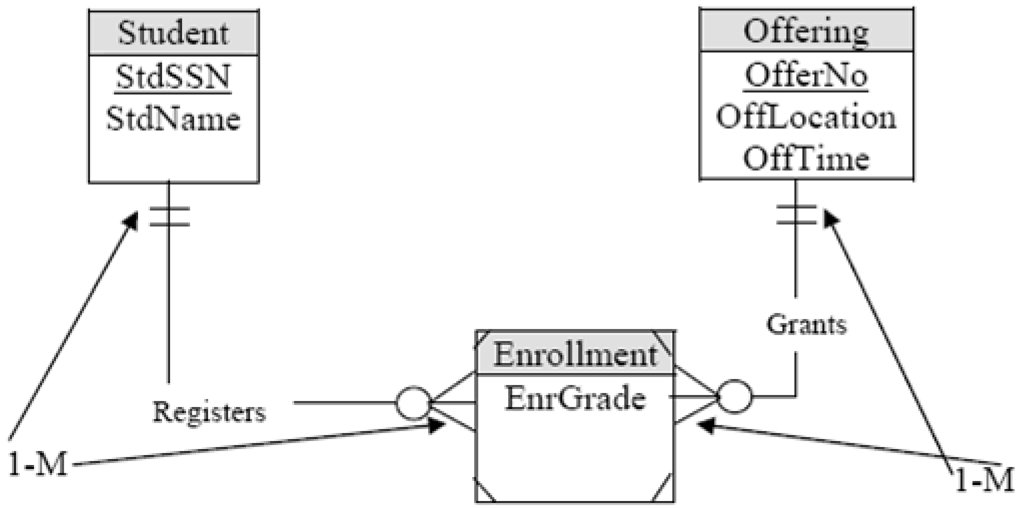


Fig x

Converting the generic SQL script to tables we have the following tables:

Student		
StdSSN (PK)	StdName	...
...	...	...

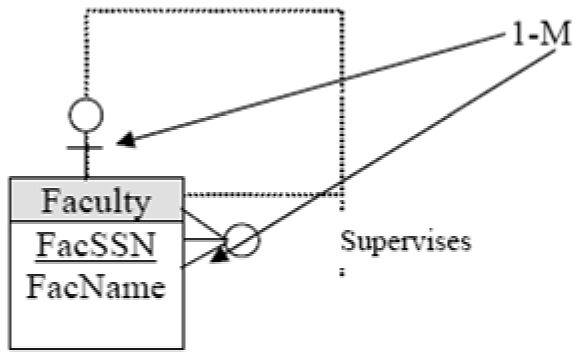
Offering				
OfferNo (PK)	OffLocation	**CourseNo** (FK)	OffTime	...
...	...	...	...	...

Enrollment				
OfferNo (PK + FK)	**StdSSN** (PK + FK)	EnrGrade	...	...
...	...	...	...	...

The rules also can be used to convert self-referencing relationships as shown in the following figures.

EXAMPLE (1-M)

(a) Manager-subordinates



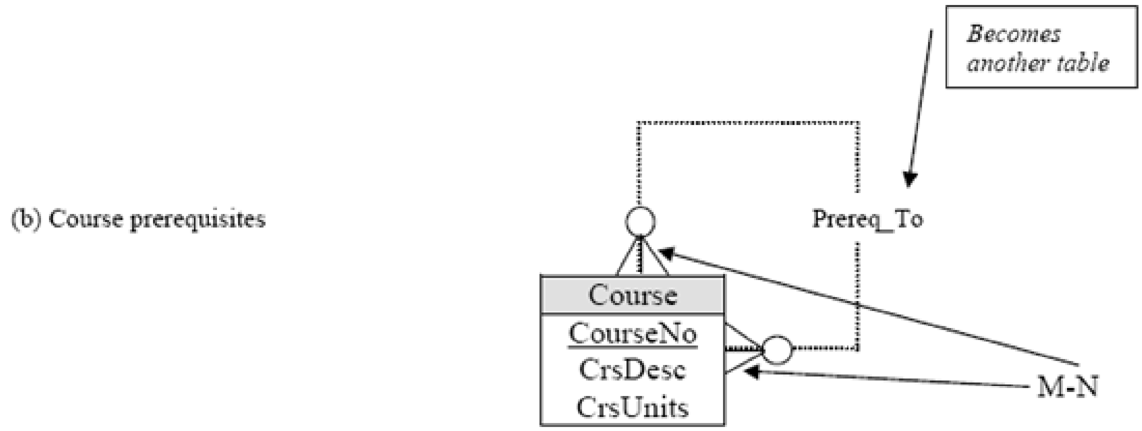
Using 1-M relationship rule, the Supervises relationship converts to a FK in the Faculty table as shown in the following SQL script.

```
CREATE TABLE Faculty(
       FacSSN              CHAR(11),
       FacName             VARCHAR(30),
       FacSupervisor       CHAR(11),
       ...
       PRIMARY KEY (FacSSN),
       FOREIGN KEY (FacSupervisor) REFERENCES Faculty (FacSSN),
);
```

Converting the generic SQL script to tables we have the following:

Faculty				
FacSSN (PK)	FacName	**FacSupervisor** (FK to the same table)	...	...
...	...	...	...	...

EXAMPLE (M-N)



Using M-N relationship rule, the **Prereq_To** relationship converts to the **Prereq_To** table with a combined PK of the course number of the prerequisite course and the course number of the dependent course.

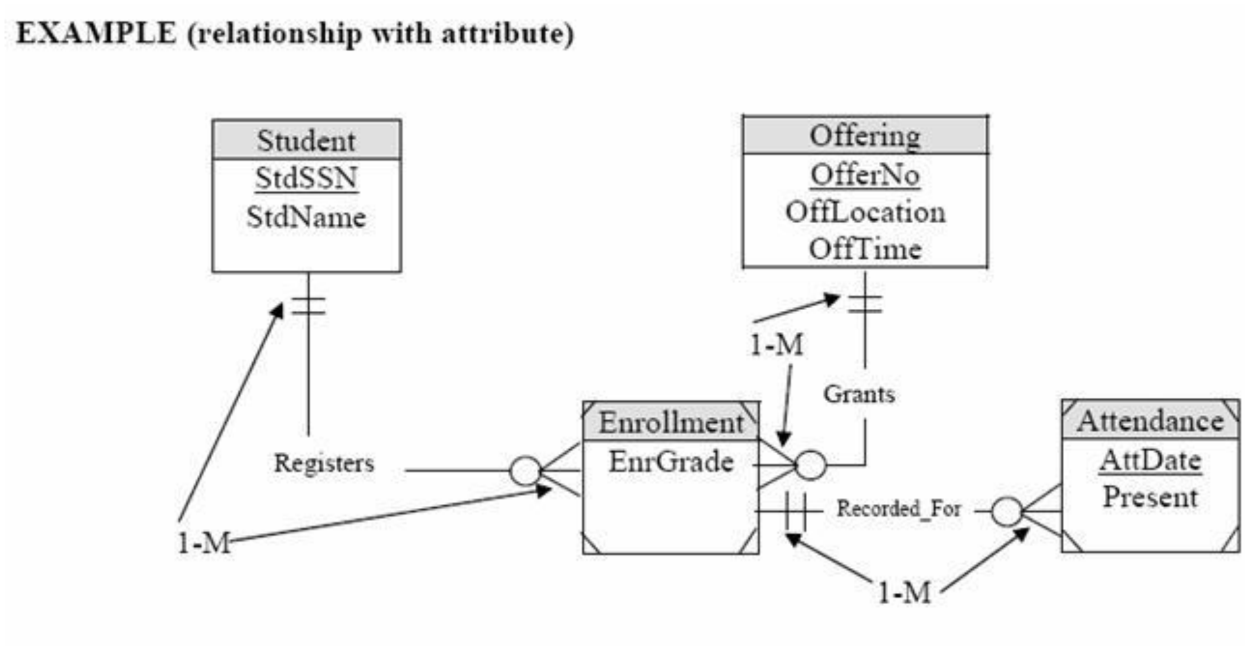
```
CREATE TABLE Course(
            CourseNo       CHAR(6),
            CrsDesc        VARCHAR(30),
            CrsUnits       SMALLINT,
            PRIMARY KEY (CourseNo)
);

CREATE TABLE Prereq_To(
          PrereqCNo     CHAR(6),
          DependCNo     CHAR(6),
          PRIMARY KEY (PrereqCNo, DependCNo),
          FOREIGN KEY (PrereqCNo) REFERENCES Course (CourseNo),
          FOREIGN KEY (DependCNo) REFERENCES Course (CourseNo)
);
```

So the M-N becomes 1-M relationships. Converting the generic SQL script to tables we have the following:

Course			
CourseNo (PK)	CrsDesc	CrsUnits	…
…	…	…	…

Prereq_To			
PrereqCNo (PK + FK)	**DependCNo** (PK + FK)	…	…
…	…	…	…

The following example shows conversion rules applied to more complex identification dependencies.



The first part of the conversion is identical to the conversion of the previous one. Application of the 1-M rule makes the combination of StdSSN and OfferNo FKs in the Attendance table as shown in the following SQL script. Note that the FKs in Attendance refer to Enrollment, not to Student and Offering. Finally, one application of the identification dependency rule makes the combination of StdSSN, OfferNo and AttDate the PK of the Attendance table.

```
CREATE TABLE Attendance(
            OfferNo           INTEGER,
            StdSSN            CHAR(11),
            AttDate           DATE,
            Present           BOOLEAN,
            PRIMARY KEY (OfferNo, StdSSN, AttDate),
            FOREIGN KEY (OfferNo, StdSSN) REFERENCES Enrollment
            (OfferNo, StdSSN)
);
```

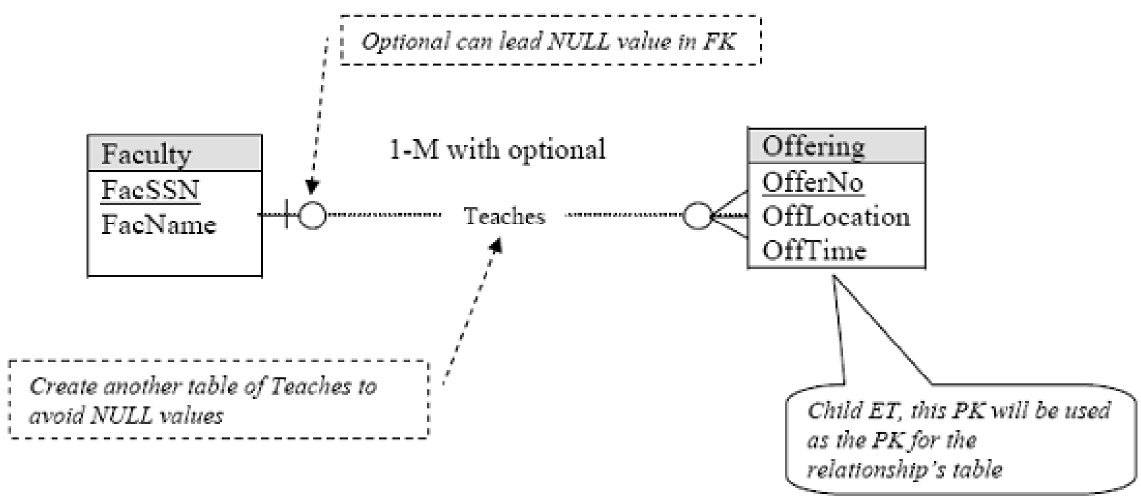
Converting the generic SQL script to tables we have the following:

Attendance			
OfferNo (PK + FK)	**StdSSN** (PK + FK)	**AttDate** (PK)	Present
…	…	…	…

Converting Optional 1-M Relationships

When you use the 1-M relationship rule for optional relationships, the resulting FK may contain NULL values. Recall that a relationship with a minimum cardinality of 0 is optional. For example the Teaches relationship in the following figure is optional to Offering because an Offering ET can be stored without being related to a Faculty ET. Converting the following figure results in two tables (Faculty and Offering) as well as a FK (FacSSN) in the Offering table.

EXAMPLE (1-M with optional minimum)



The FK should allow NULL values because the minimum cardinality of the Offering ET in the relationship is optional (may be or can be 0). However, NULL values can lead to complications in evaluating the query results. To avoid NULL values when converting an optional 1-M relationship, you can apply Rule 5 to **convert an optional 1-M relationship into a table instead of a FK**. The following SQL script shows an application of Rule 5 to the ERD of the previous figure. The Teaches table contains the

FKs OfferNo and FacSSN with NULL values not allowed for both columns. In addition the Offering table no longer has a FK referring to the Faculty table.

```
CREATE TABLE Faculty(
            FacSSN         CHAR(11),
            FacName       VARCHAR(30),
            ...
            PRIMARY KEY (FacSSN)
);

CREATE TABLE Offering(
            OfferNo        INTEGER,
            OffLocation    VARCHAR(30),
            OffTime        TIMESTAMP,
            ...
            PRIMARY KEY (OfferNo)
);

CREATE TABLE Teaches(
            OfferNo        INTEGER,
            FacSSN         CHAR(11) NOT NULL,
            -- The PK used is from child
            PRIMARY KEY (OfferNo),
            FOREIGN KEY (FacSSN) REFERENCES Faculty (FacSSN),
            FOREIGN KEY (OfferNo) REFERENCES Offering (OfferNo)
);
```

Converting the generic SQL script to tables we have the following:

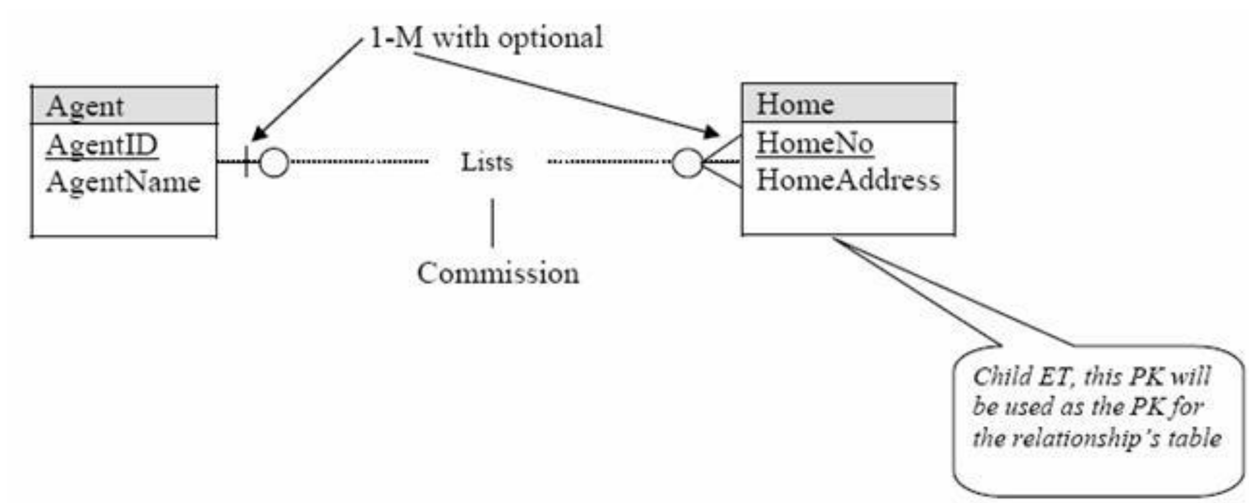
Faculty		
FacSSN (PK)	FacName	…
…	…	…

Offering			
OfferNo (PK)	OffLocation	OffTime	…
…	…	…	…

Teaches		
OfferNo (PK + FK)	**FacSSN** (FK)	...
...	NOT NULL	...

The following is another example converting 1-M relationship with an attribute. Note that the Lists table contains the Commission column.

EXAMPLE




```
CREATE TABLE Agent(
            AgentId      CHAR(10),
            AgentName    VARCHAR(30),
            ...
            PRIMARY KEY (AgentId)
);

CREATE TABLE Home(
            HomeNo        INTEGER,
            HomeAddress   VARCHAR(50),
            ...
            PRIMARY KEY (HomeNo)
);
```

```sql
CREATE TABLE Lists(
            -- NOT NULL is not used because it is PK for the table
            -- and it is already NOT NULL
            HomeNo      INTEGER,
            -- Must put NOT NULL
            AgentId     CHAR(10)      NOT NULL,
            Commission  DECIMAL(10,2),
            -- The PK used is from child
            PRIMARY KEY (HomeNo),
            FOREIGN KEY (AgentId) REFERENCES Agent (AgentId),
            FOREIGN KEY (HomeNo) REFERENCES Home (HomeNo)
);
```

Converting the generic SQL script to tables we have the following:

Agent		
AgentId (PK)	AgentName	…
…	…	…

Home		
HomeNo (PK)	HomeAddress	…
…	…	…

List		
HomeNo (PK + FK)	**AgentId** (FK)	Commission
…	NOT NULL	…

Then, from the previous example we have the fifth rule:

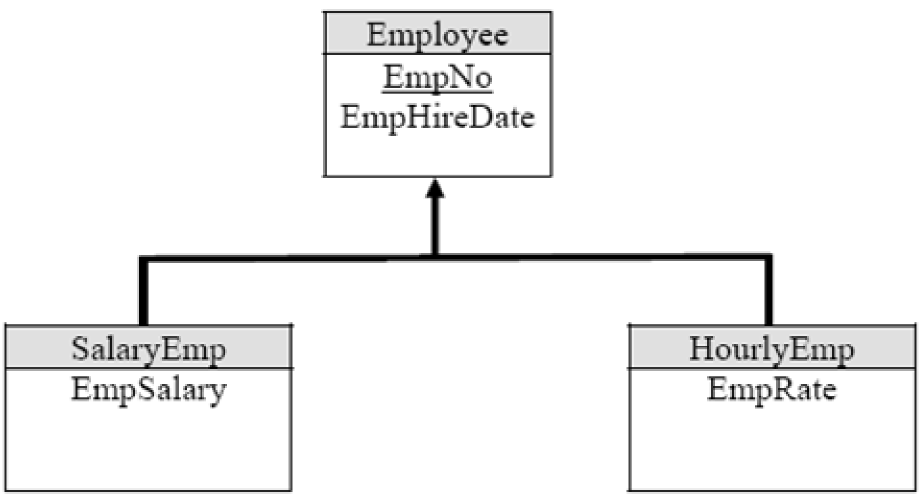
5. **Optional 1-M relationship Rule:** Each 1-M relationship with 0 for the minimum cardinality on the parent side becomes a new table. **The PK of the new table is the PK of the ET on the child (many) side of the relationship**. The new table contains FKs for the PKs of both ETs participating in the relationship. **Both FKs in the new table do not permit NULL values**. The new table also

contains the attributes of the optional 1-M relationship as in the previous example, a Commission.

Rule 5 is controversial. Using Rule 5 in place of Rule 2 (1-M Relationship Rule) avoids NULL values in FKs. However, the use of **Rule 5 results in more tables**. Query formulation can be more difficult with additional tables. In addition, query execution can be slower due to extra joins. The choice of using Rule 5 in place of Rule 2 depends on the importance of **avoiding NULL values versus avoiding extra tables**. In many database, avoiding extra tables may be more important than avoiding NULL values.

Converting Generalization Hierarchies

The approach to convert generalization hierarchies mimic the entity relationship notation as mush as possible. **Rule 6 convert each ET of a generalization hierarchy into a table**. The only column appearing that are different from attributes in the associated ERD is the **inherited PK**. In the following figure, EmpNo is a column in the SalaryEmp and HourlyEmp tables because it is the PK of the parent ET (Employee). In addition the SalaryEmp and HourlyEmp tables have a **FK** constraint referring to the Employee table. The CASCADE delete option is set in both FK constraints.




```
CREATE TABLE Employee(
            EmpNo                   INTEGER,
            EmpName                 VARCHAR(30),
            EmpHireDate             DATE,
            PRIMARY KEY (EmpNo)
);
```

```
CREATE TABLE SalaryEmp(
            EmpNo             INTEGER,
            EmpSalary         DECIMAL(10,2),
            PRIMARY KEY (EmpNo),
            FOREIGN KEY (EmpNo) REFERENCES Employee (EmpNo)
            ON DELETE CASCADE
);

CREATE TABLE HourlyEmp(
            EmpNo              INTEGER,
            EmpRate            DECIMAL(10,2),
            PRIMARY KEY (EmpNo),
            FOREIGN KEY (EmpNo) REFERENCES Employee (EmpNo)
            ON DELETE CASCADE
);
```

Converting the generic SQL script to tables we have the following:

Employee			
EmpNo (PK)	EmpName	EmpHireDate	...
...	...	...	...

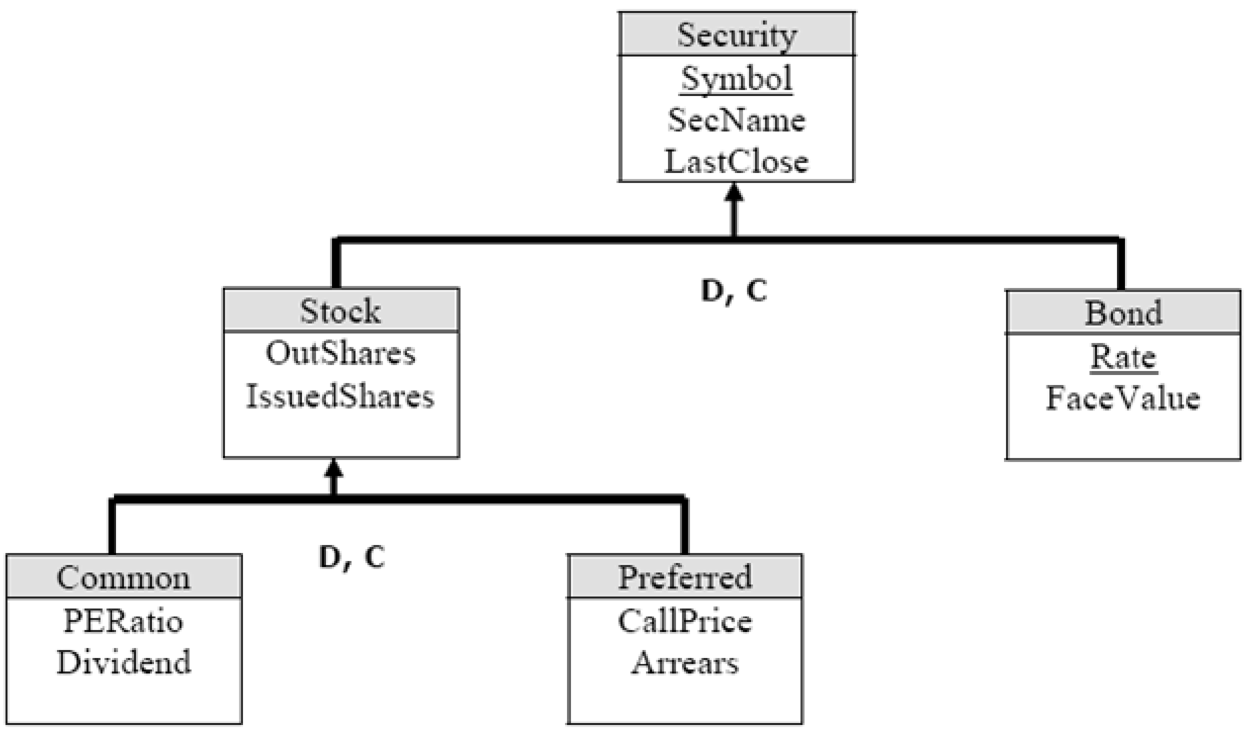
SalaryEmp		
EmpNo (PK + FK)	EmpSalary	...
shared or inherited PK, FK is ON DELETE CASCADE	...	...

HourlyEmp		
EmpNo (PK + FK)	EmpRate	...
Shared or inherited PK, FK is ON DELETE CASCADE	...	...

Well, let define rule no. 6:

6. **Generalization Hierarchy Rule:** Each ET of a generalization hierarchy becomes a table. The columns of a table are the attributes of the corresponding ET plus the PK of the parent ET. For each table representing a subtype, define a FK constraint that references the table corresponding to the parent ET. Use CASCADE (ON DELETE CASCADE) option for deletion of referenced rows.

Rule 6 also applies to generalization hierarchies of more than one level. To convert the generalization hierarchy of the following figure, five tables are produced as shown in the following SQL script. In each table, the PK of the parent (security) is included. In addition, FK constraints are added in each table corresponding to a subtype.



```
CREATE TABLE Security(
            Symbol        CHAR(6),
            SecName       VARCHAR(30),
            LastClose     DECIMAL(10,2),
            PRIMARY KEY (Symbol)
);

CREATE TABLE Stock(
            Symbol           CHAR(6),
            OutShares        INTEGER,
            IssuedShares     INTEGER,
            PRIMARY KEY (Symbol),
            FOREIGN KEY (Symbol) REFERENCES Security ON DELETE
CASCADE
);

CREATE TABLE Bond(
            Symbol        CHAR(6),
            Rate          DECIMAL(12,4),
            FaceValue     DECIMAL(10,2),
            PRIMARY KEY (Symbol),
            FOREIGN KEY (Symbol) REFERENCES Security (Symbol)
            ON DELETE CASCADE
);

CREATE TABLE Common(
            Symbol          CHAR(6),
            PERatio         DECIMAL(12,4),
            Dividend        DECIMAL(10,2),
            PRIMARY KEY (Symbol),
            FOREIGN KEY (Symbol) REFERENCES Stock (Symbol)
            ON DELETE CASCADE
);

CREATE TABLE Preferred(
            Symbol           CHAR(6),
            CallPrice        DECIMAL(12,2),
            Arrears          DECIAML(10,2),
            PRIMARY KEY (Symbol),
            FOREIGN KEY (Symbol) REFERENCES Stock (Symbol)
            ON DELETE CASCADE
);
```

Converting the generic SQL script to tables we have the following:

Security			
Symbol (PK)	SecName	LastClose	...
...	...	...	...

Stock			
Symbol (PK + FK)	OutShares	IssuedShares	...
shared PK, FK is ON DELETE CASCADE	...	...	...

Bond			
Symbol (PK + FK)	Rate	FaceValue	...
shared PK, FK is ON DELETE CASCADE	...	...	...

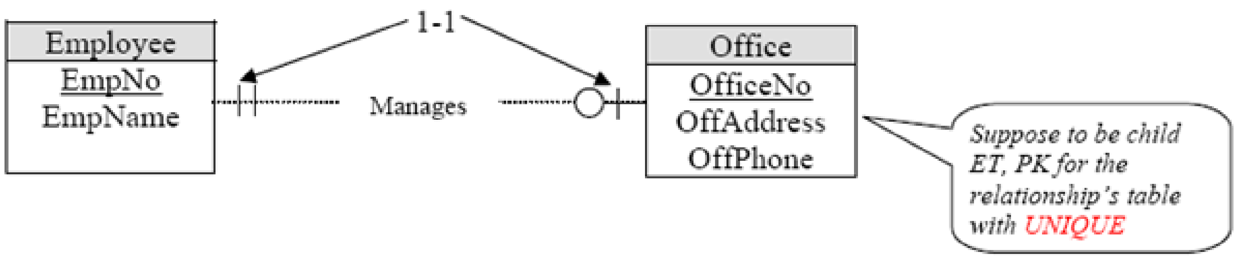
Common			
Symbol (PK + FK)	PERatio	Dividend	...
inherited PK, FK is ON DELETE CASCADE	...	...	...

Preferred			
Symbol (PK + FK)	CallPrice	Arrears	...
inherited PK, FK is ON DELETE CASCADE	...	...	...

Because the Relational Model does not directly support generalization hierarchies, there are several other ways to convert generalization hierarchies. The other approaches vary depending on the number of tables and the placement of inherited columns. Rule 6 may result in extra joins to gather all data about an entity, but there are no NULL values and only small amounts of duplicate data. For example, to collect all data about a common stock, you should join the Common, Stock and Security tables. **Other conversion approaches may require fewer joins, but result in more redundant data and NULL values.**

Converting 1-1 Relationships

Outside of generalization hierarchies, 1-1 relationships **are not common**. **They can occur when entities with separate identifiers are closely related**. For example, the following figure shows the Employee and Office ETs connected by a 1-1 relationship.



Separate ETs seem intuitive, but 1-1 relationship connects the ETs. Rule 7 converts 1-1 relationships into 2 FK unless many NULL values will results. **From the figure, most employees will not manage offices.** Thus, the conversion in the following SQL script eliminates the FK (OfficeNo) in the employee table.

```
CREATE TABLE Employee(
            EmpNo        INTEGER,
            EmpName      VARCHAR(30),
            PRIMARY KEY (EmpNo)
);

CREATE TABLE Office(
            OfficeNo       INTEGER,
            OffAddress     VARCHAR(30),
            OffPhone       CHAR(10),
            EmpNo          INTEGER,
            PRIMARY KEY (OfficeNo),
            FOREIGN KEY (EmpNo) REFERENCES Employee (EmpNo),
            UNIQUE (EmpNo)
);
```

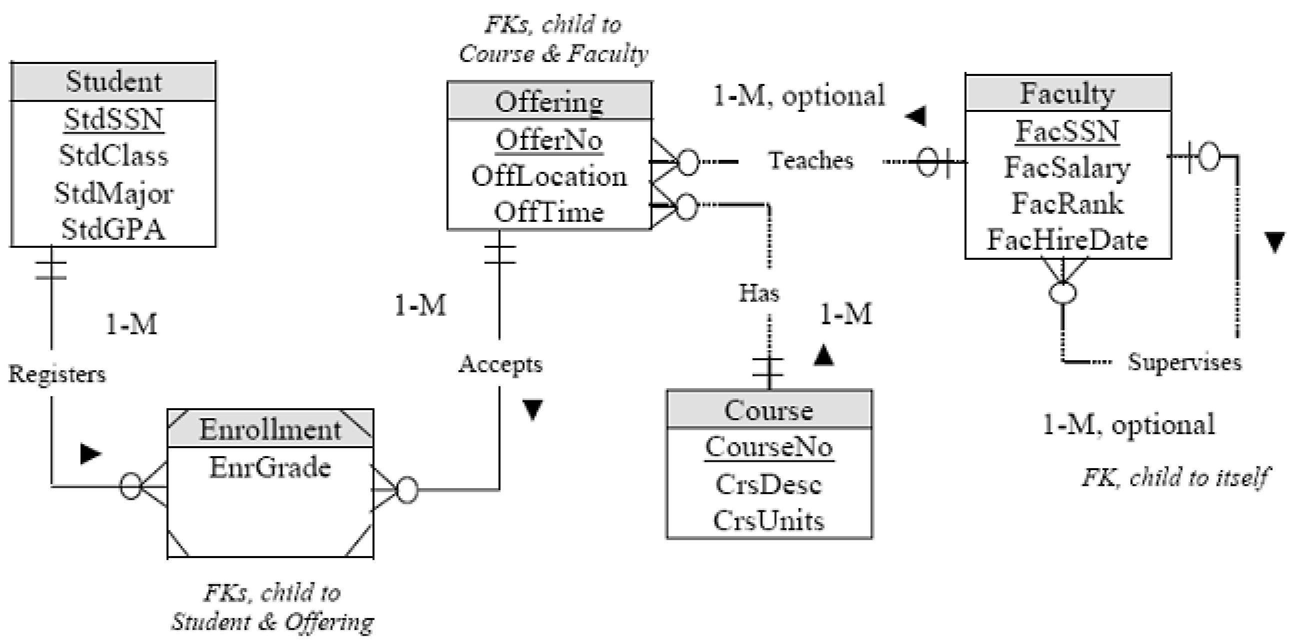
Converting the generic SQL script to tables we have the following:

Employee		
EmpNo (PK)	EmpName	...
...	...	...

Office				
OfficeNo (PK)	OffAddress	OffPhone	**EmpNo** (FK)	...
...	...	...	UNIQUE	...

7. **1-1 Relationship Rule:** Each 1-1 relationship is converted into 2 FKs. If the relationship is optional with respect to one of the ETs, the corresponding FK may be dropped to eliminate NULL values.

THE WHOLE ERD SAMPLE FOR THE PREVIOUS EXAMPLE



```
create table Student(
stdSSN                  char(11) not null,
stdFirstName            varchar(30) not null,
stdLastName             varchar(30) not null,
stdCity                 varchar(30) not null,
stdState                char(2) not null,
stdZip                  char(10) not null,
stdMajor                char(6),
stdClass                char(2),
stdGPA                  numeric(3,2),
PRIMARY KEY (StdSSN)
);

create table Course(
CourseNo         char(6) not null,
crsDesc          varchar(50) not null,
CrsUnits         integer,
PRIMARY KEY (CourseNo)
);



create table offering(
OfferNo           INTEGER not null,
CourseNo          char(6) not null,
OffTerm           char(6) not null,
OffYear           INTEGER not null,
OffLocation       varchar(30),
OffTime           varchar(10),
FacSSN            char(11),
OffDays           char(4),
PRIMARY KEY (OfferNo),
FOREIGN KEY (CourseNo) REFERENCES Course(CourseNo),
FOREIGN KEY (FacSSN) REFERENCES Faculty(FacSSN)
);

create table Faculty(
FacSSN                  char(11) not null,
FacFirstName            varchar(30) not null,
FacLastName             varchar(30) not null,
FacCity                 varchar(30) not null,
FacState                char(2) not null,
FacZipCode              char(10) not null,
FacRank                 char(4),
```

```
FacHireDate             date,
FacSalary               numeric(10,2),
FacSupervisor           char(11),
FacDept                 char(6),
PRIMARY KEY (FacSSN),
FOREIGN KEY (FacSupervisor) REFERENCES
Faculty(FacSupervisor)
);

create table Enrollment(
OfferNo                  INTEGER not null,
StdSSN                   char(11) not null,
EnrGrade                 numeric(3,2),
PRIMARY KEY (OfferNo, StdSSN),
FOREIGN KEY (OfferNo) REFERENCES Offering
ON DELETE CASCADE,
FOREIGN KEY (StdSSN) REFERENCES Student(StdSSN) ON DELETE
CASCADE
);
```