**CSCI 4333.1**

**9/23/2024**

**Transforming UML Class Diagrams to Relational Models**

by K. Yue

**1. Transforming OO Model to the Relational Model**

* Once a *conceptual* OO data model is constructed, it needs to be mapped for implementation in the selected *logical* database model. See [IntroDataModeling.html](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/model/IntroDataModeling.html).
* If a relational DB is used, the mapping will be from OO (classes, attributes, associations, etc.) to relational schema (relations, attributes, keys, etc.)
* Note that the relational model and the OO model are *very different*, even though diagrams representing the two models look similar (there are only a finite number of common suitable shapes).
* Computer-Aided Software Engineering (CASE) tools or DB modeling tools may provide varying degrees of facilities for automatic generating relational schema and corresponding SQL statements.
* However, it is important to understand the mechanism that a tool uses to generate the relational schema and make adjustment if needed.

**1.1 Model Transformation**

The problem is: Source Model --> Target Model.

* Source Model: UML: OO Model
  1. Basic elements:
     1. Class
     2. Attribute: can be multi-valued.
     3. Association
  2. Secondary elements:
     1. Object
     2. Multiplicity
     3. Data type
     4. Default value
     5. Constraint
     6. Stereotype for RDB extensions: e.g., candidate keys, primary keys, unique, derived, nullability, etc.
     7. ...
* Target Model: Relational Model
  1. Basic elements:
     1. Relation
     2. Attribute (column/field): It should be single-valued (atomic). Otherwise, first normal form (1NF) is not satisfied.
  2. Secondary elements:
     1. Row (tuple)
     2. Data type
     3. Nullability
     4. Constraint
     5. Candidate key
     6. Primary key
     7. Foreign key
     8. Index
     9. ...
* There are only two basic elements in the targeted model to consider. Data in the relational model can be stored only in two ways:
  1. relation:
     1. more flexible,
     2. can hold attributes to store a collection of logically related data
     3. more complex.
  2. attribute:
     1. should be single-valued (if good design, i.e., the first normal form, is to be assured.)
     2. Simple
     3. Thus, if attributes are sufficient in model transformation, they are preferred.
* Different RDBMS provide different features.
  + 1. Thus, the targeted RDB model is not universal.
    2. It is necessary to define vendor-specific transformation rules.

**2. Transformation Rules**

* We present a set of mapping rules below. It is not meant to be complete or universal.
* Examples of comparable transformation rules:
  + A relative simple one: <http://web.fe.up.pt/~ssn/2010/lbaw/slides/lbaw-uml2rel.eng.pdf> a
  + A more elaborated one based on agile methodology: [http://www.agiledata.org/essays/mappingObjects.html](http://www.agiledata.org/essays/mappingObjects.html#Figure2IncludingShadowInformation).
* Do not mechanically follow these rules. Instead, understand the rationale behind the rules and adapt.
* All OO model details should be implemented in the targeted model in some ways.
  + database level: preferred.
  + middle layer level
  + application level

**2.1 Classes**

**C1**. A class C is mapped or transformed to a relation RC.

1. Relations may later be merged and/or reorganized in design refinement and performance tuning.
2. The relation may use the same name as the class.
3. As a result, all infomration of an object of class C is stored as a row in RC.

**Rationale:**

1. A class is a logical unit for encapsulating related data and a relation has the same property.

A yellow box with black text

Description automatically generated

|  |  |
| --- | --- |
| 2 | TRMember(TRMemberId, LName, FName, ScreenName, StartTime, EMail, ReferrerTRMemberId, Username) |
| Candidate Keys | [1] TRMemberId, [2] ScreenName |
| Foreign Keys | [1] ReferrerTRMemberId references TRMember(TRMemberId), [2] Username references Account(Username) |
| Nullable Attributes | TRRefererMemberId, EMail |
| Non-nullable Attributes | TRMemberId, LName, FName, ScreenName, StartTime |
| Notes |  |

**2.2 Attributes**

**Basic:**

**ATT1. Single-Valued Attributes.**Map all *single*-valued attributes (with simple data types) of a class C as attributes of RC, the relation for the class C.

**ATT2**. **Multi-Valued Attributes.** For *each* *multi-valued* attribute A, Email of the class C, student, create a *new* relation RCA, StudentEMail containing the attribute A, EMail and the primary key, RCId, STuId, of the relation RC, student (which implements the class C).

1. (RCId, A) is a composite candidate key.
2. RCId, stuId, is a foreign key referencing RC(RCId), Student(StuId)
3. A surrogate key, such as RCA\_Id, may be created to serve as the *simple* primary key.
4. The name of RCA should be meaningfully selected.

A yellow square with black text

Description automatically generated

Student(LName, …)

Example student object S1:

LName: Swift:  
 Emails: [swift@uhcl.edu](mailto:swift@uhcl.edu), [swift@gamil.com](mailto:swift@gamil.com)  
 Orgs: CS, IEEE, ACM

Student table:

|  |  |  |  |
| --- | --- | --- | --- |
| **LName** | **StuID (PK)** | **~~Emails? Multi-valued~~** |  |
| Swift | S1 | [~~swift@uhcl.edu~~](mailto:swift@uhcl.edu)~~,~~ [~~swift@gamil.com~~](mailto:swift@gamil.com) |  |
|  |  |  |  |

OK?

A yellow box with black text

Description automatically generated

Student table:

|  |  |  |  |
| --- | --- | --- | --- |
| **LName** | **StuID (PK)** | **Email1** | Email2 |
| Swift | S1 | [swift@uhcl.edu](mailto:swift@uhcl.edu) | [swift@gamil.com](mailto:swift@gamil.com) |
|  |  |  |  |

Instead, new table:

StudentEMail

|  |  |  |
| --- | --- | --- |
| **StuId (FK = Student(StuId)** | **EMail** | SE\_Id |
| S1 | [swift@uhcl.edu](mailto:swift@uhcl.edu) | 1 |
| S1 | [swift@gamil.com](mailto:swift@gamil.com) | 2 |
| S2 | [swift@gamil.com](mailto:swift@gamil.com)? Ok? | 3 |

StudentOrg

|  |  |  |
| --- | --- | --- |
| **StuId (FK)** | **Org** |  |
| S1 | CS |  |
| S1 | IEEE |  |
| S1 | ACM |  |
|  |  |  |

**Secondary:**

**ATT3.** **Zero in Multiplicity.** If the multiplicity of an attribute is specified, to handle the case of 0:

1. If 0 is allowed in the UML model (e.g., 0..1, 0..\* in the UML class diagram), the attribute is nullable. Add the NULL specifier in the column definition in the RDBMS. (NULL is usually the default)
2. If 0 is not allowed, add the NOT NULL specifier in the column definition.

**ATT4. Default Values.**The default value of an attribute can be directly implemented in SQL DDL.

**ATT5. Data Types.**Data type mapping should be handled adequately, effectively, and consistently.

1. In later modeling phases, one may use SQL data types of the targeted DBMS in the class diagram.
2. If available, consider using user-defined data types in the targeted DBMS.

***Example:***

UML for toyu.student:

A diagram of a student

Description automatically generated

Relational schema (in HW assignment format):

|  |  |
| --- | --- |
| **Relation** | Student(*StuId, fname, lname*, major, minor, *ach*, advisor) |
| Candidate Keys | [1] StuId |
| Foreign Keys | [1] major references department(deptCode), [2] minor references department(deptCode), [3] advisor references faculty(facId) |
| Nullable Attributes | major, minor, advisor, *ach* |
| Non-nullable Attributes | stuId, fname, lname |
| Notes |  |

Relational schema in SQL (with more implementation details):

CREATE TABLE IF NOT EXISTS Student    (  
    stuId        INT NOT NULL,  
    fname        VARCHAR(30) NOT NULL,  
    lname        VARCHAR(30) NOT NULL,  
    major        CHAR(4) NULL,  
    minor        CHAR(4) NULL,  
   -- ach: accumulated credit hours, including transferred credits.  
    ach          INTEGER(3) NULL DEFAULT 0,  
    advisor      INT NULL,  
    CONSTRAINT Student\_stuId\_pk PRIMARY KEY(stuId),  
   -- an artificial example of a CHECK constraint.  
    CONSTRAINT Student\_ach\_cc CHECK ((ach>=0) AND (ach < 250)),  
    CONSTRAINT Student\_major\_fk FOREIGN KEY (major)  
        REFERENCES Department(deptCode) ON DELETE CASCADE,  
    CONSTRAINT Student\_minor\_fk FOREIGN KEY (minor)  
        REFERENCES Department(deptCode) ON DELETE CASCADE,  
    CONSTRAINT Student\_advisor\_fk FOREIGN KEY (advisor)  
        REFERENCES Faculty(facId)  
);

***Example:***

Multi-valued attributes: consider the class Member with the following attributes:

1. Member\_Id: <<PK>>
2. Screen\_Name <<unique>>
3. Hobbies[0..\*]
4. Medals[0..\*]

A yellow card with black text

Description automatically generated

Reasonable relation schema: three relations used

1. Member(MemberId, ScreenName):
   1. CK: [1] MemberId, [2] *ScreenName*
2. Hobby(HobbyId, MemberId, Hobby):
   1. CK: [1] HobbyId, [2] MemberId, Hobby
   2. FK: [1] MemberId references Member(MemberId)
   3. A surrogate key, HobbyId, is created as the primary key.
3. Medal(MedalId, MemberId, Medal):
   1. CK: [1] MedalId, [2] MemberId, Medal.
   2. FK: [1] MemberId references Member(MemberId)
   3. A surrogate key, MedalId, is created as the primary key.

All columns in the tables above are not nullable.

**ATT6. Single-Valued Composite Data Types.** A single-valued attribute of a *composite* data type (such as set, list, array) can be mapped in various ways. E.g. JSON

1. If there is a comparable composite data type in the targeted DBMS, it can be implemented as an attribute of that data type in the relation.
   * The relation will no longer be in the first normal form.
   * Care should be taken in handling the difference in data type mapping.
2. Otherwise, regard the attribute as a multi-valued attribute and apply rule ATT2.

**ATT7. Derived Attributes.** For a derived attribute A: no independent variable.

E.g. Student(StuId, LName, FName, Dob, Age, ….)

1. It can be implemented and stored as an attribute of the relation.
   * Mechanisms, such as triggers or stored procedures, should be used to ensure data consistency. The derived column should be consistent with the data that derives its value.
2. It may not be stored as a column directly in any relation.
   * Mechanisms, such as virtual columns, views or stored functions, may be used to provide standard access to the derived attributes.

***Example:***

A class Rectangle has three attributes:

* Length
* Width
* \Area or <<derived>>: derived.

A rectangular yellow rectangle with black text

Description automatically generated

What may the relational schema look like?

One solution:

Rectangle(RectangleId, Length, Width)

with a view Rect define as

SELECT DISTINCT RectangleId, Length, Width, Length \* Width as Area  
FROM Rectangle;

Alternatively, use*virtual* columns: a column that is computed in real-time by an expression and not stored.

CREATE or replace TABLE rectangle (  
  width DOUBLE,  
  height DOUBLE,  
  area DOUBLE AS (width \* height) virtual  
);

**2.3 Keys and Constraints**

**KC1. Primary Key Identified.** If a relation R implements a class C or an association (class) AC, and C or AC has identified the PK, set it as the primary key of R.

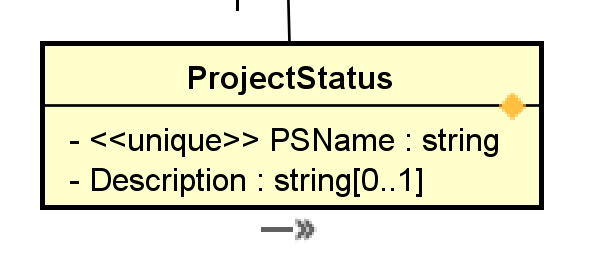
A yellow box with black text

Description automatically generated

|  |  |
| --- | --- |
| 2 | TRMember(TRMemberId, LName, FName, ScreenName, StartTime, EMail, ReferrerTRMemberId, Username) |
| Candidate Keys | [1] TRMemberId, [2] ScreenName |
| Foreign Keys | [1] ReferrerTRMemberId references TRMember(TRMemberId), [2] Username references Account(Username) |
| Nullable Attributes | TRRefererMemberId, EMail |
| Non-nullable Attributes | TRMemberId, LName, FName, ScreenName, StartTime |
| Notes |  |

**KC2. Candidate Keys.** If a relation R implements a class C or an association (class) AC, and C or AC has candidate keys K's, set **all** K's as candidate keys of R.

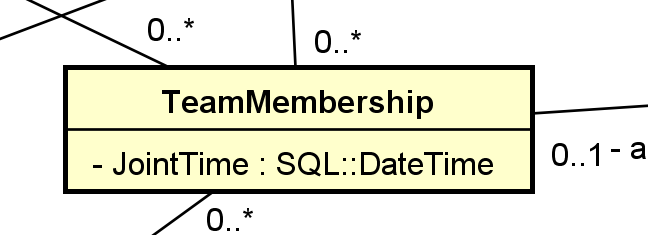
* If no primary key has been identified, select a CK to serve as the primary key, and apply KC1.
* A candidate key can be implemented by using the 'unique' and non-null constraint together in SQL.

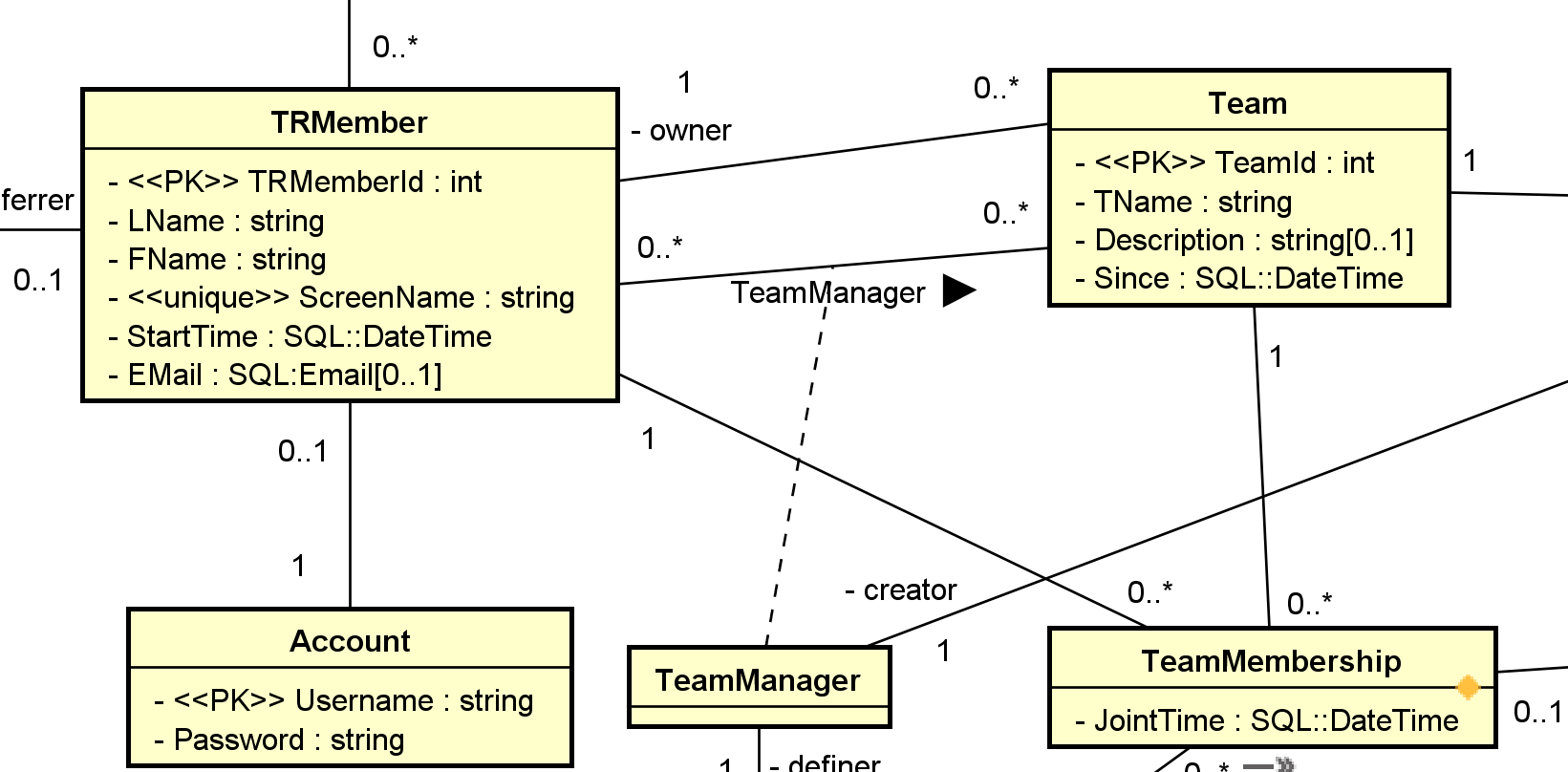


|  |  |
| --- | --- |
| 11 | ProjectStatus(PSId, PSName, Description) |
| Candidate Keys | [1] PSId, [2] PSName |
| Foreign Keys |  |
| Nullable Attributes | Description |
| Non-nullable Attributes | PSId, PSName |
| Notes | [1] PSId is created as a surrogate primary key. |

**KC3. No Primary Key Identified.**If a relation R implements a class C or an association (class) A, and C or AC has no candidate key, create a *surrogate* primary key K for R.

1. This is needed as every relation must have at least one candidate key.





|  |  |
| --- | --- |
| 5 | TeamMember(TMId, TRMemberId, TeamId, JointTime) |
| Candidate Keys | [1] TMId, [2] TRMemberId, TeamId, JointTime |
| Foreign Keys | [1] TRMemberId references TRMember(TRMemberId), [2] TeamId references Team(TeamId) |
| Nullable Attributes |  |
| Non-nullable Attributes | TMId, TRMemberId, TeamId, JointTime |
| Notes | [1] TMId is created as a surrogate primary key. |

***Example:***

for the class Department in toyu:

A yellow sign with black text

Description automatically generated

Relational schema (in HW assignment format):

|  |  |
| --- | --- |
| **3** | Department(deptCode, deptName, schoolCode, numStaff) |
| Candidate Keys | [1] deptCode, [2] deptName |
| Foreign Keys | [1] schoolCode references School(schoolCode) |
| Nullable Attributes | schoolCode, numStaff |
| Non-nullable Attributes | deptCode, deptName |
| Notes |  |

Relational schema in SQL (with more implementation details):

CREATE TABLE IF NOT EXISTS Department (  
    deptCode    CHAR(4) NOT NULL,  
    deptName    VARCHAR(30) *NOT NULL,*  
    schoolCode  CHAR(3) NULL,  
    numStaff  TINYINT NULL,  
    CONSTRAINT Department\_deptCode\_pk PRIMARY KEY (deptCode),  
   -- alternate keys: [1] deptName    
    *CONSTRAINT Department\_name\_ck UNIQUE (deptName),*  
    CONSTRAINT Department\_schoolCode\_fk FOREIGN KEY (schoolCode)  
        REFERENCES School(schoolCode)  
);

**KC4.** For a stereotype:

1. Some may be directly implemented in SQL DDL, e.g., PK, CK, unique, etc.
2. Otherwise, it is necessary to consider where and how it is implemented.

**2.4 Associations**

**A1. Many-to-one Association.** For a many to one association A between C1 (the class with the one multiplicity) and Cm, add a column R1\_Id into the relation Rm (which implements Cm).

1. Assume that R1\_Id is the primary key of the relation R1 (which implements C1).
2. R1\_Id is a foreign key in Rm referencing R1(R1\_Id).
3. The name R1\_Id may be renamed.
4. R1\_Id is not null in Rm iff (if and only if) 0 is not allowed (i.e., 1..1) for C1.
5. Any single-valued attribute of the association is mapped to a column in Rm.
6. If the association A is an association class, single-valued attributes of A can be stored as attributes of R1.
7. If you have composite or multi-valued attributes of the relationship, you should consider promoting the association to a regular class in your UML class diagram.

***Example:***

A diagram of a team manager

Description automatically generated

|  |  |
| --- | --- |
| 5 | TeamMember(TMId, TRMemberId, TeamId, JointTime) |
| Candidate Keys | [1] TMId, [2] TRMemberId, TeamId, JointTime |
| Foreign Keys | [1] TRMemberId references TRMember(TRMemberId), [2] TeamId references Team(TeamId) |
| Nullable Attributes |  |
| Non-nullable Attributes | TMId, TRMemberId, TeamId, JointTime |
| Notes | [1] TMId is created as a surrogate primary key. |

For:

A diagram of a diagram

Description automatically generated

We have the three numbered associations implemented by the three foreign keys below.

|  |  |
| --- | --- |
| **4** | Faculty(facId, fname, lname, *deptCode*, rank) |
| Candidate Keys | [1] facId |
| Foreign Keys | [1] deptCode references Department(deptCode) |
| Nullable Attributes | rank, *deptCode* |
| Non-nullable Attributes | facId, fname, lname |
| Notes |  |
| **6** | Class(classId, *courseId*, semester, year, *facId*, room) |
| Candidate Keys | [1] classId |
| Foreign Keys | [1] courseId references Course(courseId), [2] facId references Faculty(facId) |
| Nullable Attributes | room |
| Non-nullable Attributes | classId, *courseId*, semester, year, *facId* |
| Notes |  |
|  |  |

**A2. Many-to-many Association.** For a many-to-many association, TeamManager, (including association classes) between classes CA (TRMember) and CB (Team), create a*new* relation RAB(RA\_Id, RB\_Id).

1. (RA\_Id, RB\_Id): (TRMemberId, TeamId) is a candidate key.
2. RA\_Id (TRMemberId) references RA(RA\_Id), TRMember(TRMemberId) as a foreign key.
3. RB\_Id (TeamId) references RB(RB\_Id), Team(TeamId) as a foreign key.
4. An additional surrogate key, such as RAB\_Id, ManagerId, can be created.

***Example:***

***TeamManager: many to many.***

A diagram of a team manager

Description automatically generated

|  |  |
| --- | --- |
| 2 | TRMember(TRMemberId, LName, FName, ScreenName, StartTime, EMail, ReferrerTRMemberId, Username, TeamIds? No, multivalued. |
| Candidate Keys | [1] TRMemberId, [2] ScreenName |
| Foreign Keys | [1] ReferrerTRMemberId references TRMember(TRMemberId), [2] Username references Account(Username) |
| Nullable Attributes | TRRefererMemberId, EMail |
| Non-nullable Attributes | TRMemberId, LName, FName, ScreenName, StartTime |
| Notes |  |

|  |  |
| --- | --- |
| 3 | Team(TeamId, TName, Description, Since, OwnerTRMemberId, ManagerTRMemberIds? No, multivalued.) |
| Candidate Keys | [1] TeamId |
| Foreign Keys | [1] OwnerTRMemberId references TRMember(TRMemberId) |
| Nullable Attributes | Description |
| Non-nullable Attributes | TeamId, TName, Since, OwnerTRMemberId |
| Notes |  |
| 4 | TeamManager(ManagerId, TRMemberId, TeamId) |
| Candidate Keys | [1] ManagerId, [2] TRMemberId, TeamId |
| Foreign Keys | [1] TRMemberId references TRMember(TRMemberId), [2] TeamId references Team(TeamId) |
| Nullable Attributes |  |
| Non-nullable Attributes | ManagerId, TRMemberId, TeamId |
| Notes | [1] ManagerId is created as a surrogate primary key. |

For:

A diagram of a program

Description automatically generated

We have:

|  |  |
| --- | --- |
| **8** | Enroll(stuId, classId, grade, n\_alerts) |
| Candidate Keys | [1] stuId, classId |
| Foreign Keys | [1] stuId references Student(stuId), [2] classId references Class(classId), [3] grade references Grade(grade) |
| Nullable Attributes | grade, n\_alerts |
| Non-nullable Attributes | stuId, classId |

A diagram of a group of objects

Description automatically generated

|  |  |
| --- | --- |
| 10 | TeamMemberRole(TMRId, TMId, RoleId) |
| Candidate Keys | [1] TMRId, [2] TMId, RoleId |
| Foreign Keys | [1] TMId references TeamMember(TMId), [2] RoleId references Role(RoleId) |
| Nullable Attributes |  |
| Non-nullable Attributes | TMRId, TMId, RoleId |

|  |  |
| --- | --- |
| 7 | Role(RoleId, RLNum) |
| Candidate Keys | [1] RoleId |
| Foreign Keys | [1] RLNum references RoleLevel(RLNum) |
| Nullable Attributes |  |
| Non-nullable Attributes | RoleId, RLNum |
| Notes | [1] RoleId is created as a surrogate primary key. [2] We used three relations to implement the three classes Role, UserDefinedRole and StandardRole. It is possible to use only one relation. |

|  |  |
| --- | --- |
| 5 | TeamMember(TMId, TRMemberId, TeamId, JointTime) |
| Candidate Keys | [1] TMId, [2] TRMemberId, TeamId, JointTime |
| Foreign Keys | [1] TRMemberId references TRMember(TRMemberId), [2] TeamId references Team(TeamId) |
| Nullable Attributes |  |
| Non-nullable Attributes | TMId, TRMemberId, TeamId, JointTime |
| Notes | [1] TMId is created as a surrogate primary key. |

**A3. One-to-one Association.** For a one to one association between classes CA and CB, there are several options:

1. Treat CA as C1 and CB as Cm and apply A1.
2. Treat CA as Cm and CB as C1 and apply A1.
3. Merge the two relations RA and RB into one. (In this case, you may want to refactor the class diagram.)

A diagram of a number

Description automatically generated

|  |  |
| --- | --- |
| **3** | Department(deptCode, deptName, schoolCode, numStaff) |
| Candidate Keys | [1] deptCode, [2] deptName |
| Foreign Keys | [1] schoolCode references School(schoolCode) |
| Nullable Attributes | schoolCode, numStaff |
| Non-nullable Attributes | deptCode, deptName |
| Notes |  |

**A4. N-ary Associations.** For any n-ary association (n>2), a new relation is needed.

1. You should consider using binary associations instead.
2. A ternary association can be modeled as a regular class with three binary associations with the participating classes in the ternary association.

***Example:***

Consider the ternary association between the classes Supplier, Part, and Warehouse with an association attribute quantity.

E.g. A supplier S1 supplies (shipment of) 25 part P1 to the warehouse W1.

It can reasonably be replaced by a new class and three binary associations.

A diagram of a supply chain

Description automatically generated

Supply(SupplyId, SupplierId, PartId, WarehouseId, Quantity):

1. CK: [1] SupplyId, [2] SupplierId, PartId, WarehouseId
2. FK: [1] SupplierId references Supplier(SupplierId), [2] PartId references Part(PartId), [3] WarehouseId references Warehouse(WarehouseId).
3. All attributes in Supply is not nullable.

***Example:***

Checkout the UML diagram and relation schema for [toyu](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/toyu/toyu.html) and [swim](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/notes/swim/swim.html).