**9/21/2021**

Self-annotation:

Excel’s table:

1. Data model: 2-d array.
	1. Ordered: indices.
	2. Duplicate allowing
2. no primary key
3. No domain.

RDB table:

1. Set-theoretic:
	1. Unordered
	2. No duplicated: {1,1,2,2} = {2,1}
2. primary key.
3. Relationship between the tables.
4. Easier to query by SQL.

Student table:

{stuId} is unique

{stuId, fname} is unique? Yes. Fname is not needed. Not minimal.

Superkey is unique but may or may be minimal.

{stuId, classId} is a candidate key (composite) of the table Enroll.

{stuId} is a simple CK of the table Student.

What is the key of the database toyu? Bad question.

Right Q: what are the candidate keys of the table Student? StuId.

Meaning and scope of concepts.

A foreign key of a relation is a set of attributes that is a candidate key in a parent relation.

Major is a foreign key of the student table references department(deptCode).

E.g.

**SELECT** **DISTINCT** s.stuId, d.deptName **AS** `Major subject`

**FROM** student **AS** s **INNER** **JOIN** department **AS** d

 **ON** (s.major = d.deptCode)

CREATE TABLE IF NOT EXISTS Student (

 stuId INT NOT NULL,

 fname VARCHAR(20) NOT NULL,

 lname VARCHAR(20) NOT NULL,

 major VARCHAR(4) NULL,

 minor VARCHAR(4) NULL,

 credits integer(3) DEFAULT 0,

 advisor INT NULL,

 CONSTRAINT Student\_stuId\_pk PRIMARY KEY(stuId),

 CONSTRAINT Student\_credits\_cc CHECK ((credits>=0) AND (credits < 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)

);

CREATE TABLE IF NOT EXISTS Enroll(

 stuId INT NOT NULL,

 classId INT NOT NULL,

 grade VARCHAR(2),

 n\_alerts INT,

 CONSTRAINT Enroll\_classId\_stuId\_pk PRIMARY KEY (classId, stuId),

 CONSTRAINT Enroll\_classNumber\_fk FOREIGN KEY (classId)

 REFERENCES Class(classId) ON DELETE CASCADE,

 CONSTRAINT Enroll\_stuId\_fk FOREIGN KEY (stuId)

 REFERENCES Student (stuId) ON DELETE CASCADE,

 CONSTRAINT Enroll\_grade\_fk FOREIGN KEY (grade)

 REFERENCES Grade (grade) ON DELETE CASCADE

);

1 composite PK + 3 FK.

**Classes**

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

1. Relations may later be merged in design refinement and performance tuning.
2. The relation may use the same name as the class.

**Attributes**

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

**ATT2**. (Basic) For *each* *multi-valued* attribute A of the class C, create a new relation RCA containing the attribute A and the primary key, RCId, of the relation RC (which implements the class C).

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

**ATT3.** A single-valued attribute of composite data type (such as set, list, array) can be mapped in various ways.

1. If there is an 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.

**ATT4.** For a derived attribute A: (not independent: e.g. GPA)

|  |  |  |
| --- | --- | --- |
| stuId | FName | GPA (derived: computed) |
| 11111 | Bun | 3.1 -> 4.0 (no, no) |

1. It can be implemented and stored as an attribute of the relation.
	* Mechanism, such as triggers or stored procedures, should be used to ensure data consistency. That is 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.
	* Mechanism, such as virtual columns, views or stored functions, may be used to provide standard access to the derived attributes.

**ATT5.** Data type matching should be handled 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.

**ATT6.** If the multiplicity of an attribute is specified for the case of 0.

1. If 0 is allowed, add the NULL specifier in the column definition. (NULL is usually the default)
2. If 0 is not allowed, add the NOT NULL specifier in the column definition.

**ATT7.**The default value of an attribute may be directly implemented in SQL DDL.

**Keys and Constraints**

**KC1.** If a relation R implements a class C or an association (class) A, and C or A has candidate keys K's, set **all** K's as candidate keys in R.

**KC2.**If a relation R implements a class C or an association (class) A, and C or A has no candidate key, create a surrogate candidate key SK for R.

1. This is needed as every relation must have at least one candidate key.
2. SK is usually the primary key.

**KC3.** All candidate keys can be implemented by using the 'unique' and non-null constraint in SQL.

**KC4.** Select a candidate key as the primary key and set it accordingly in the relation.

**KC5.** 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 it is implemented.

***Example:***

Consider the class Member with the following attributes:

1. Member\_Id: <<PK>>: single-valued: 123 (Bun)
2. Member\_ScreenName <<unique>>: single-valued
3. Hobby\*: multi-valued (hobbies: string[0..\*]): eat, sleep, drink
4. Medal\* <<ordered>>; zinc, aluminum, mercury

Member Table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Member\_Id** | **Scrren\_Name** | **Hobbies (no, no: not in 1NF)** | **Medals** |
| 123 | Bun | eat, sleep, drink | zinc, aluminum, mercury |
|  |  |  |  |



What may the relational schema look like?

1. Member(MemberId, MemberScreenName):
	1. CK: [1] MemberId, [2] MemberScreenName

Member:

|  |  |
| --- | --- |
| **Member\_Id** | **Scrren\_Name** |
| 123 | Bun |
| 222 | Sooraj |
| 333 | Preethi |

1. Hobbies(HobbiesId, MemberId, Hobby):
	1. CK: [1] HobbiesId, [2] {MemberId, Hobby}
	2. FK: [1] MemberId references Member(MemberId)
	3. A surrogate key, HobbiesId, is created as the primary key.

Hobby:

|  |  |  |
| --- | --- | --- |
| **Member\_Id (FK)** | **Hobby** | **HobbiesId (optional: for simple PK)** |
| 123 | Eat | 1 |
| 123 | Sleep | 2 |
| 123 | Drink | 3 |
| 222 | Sleep | 4 |
| 333 | Sleep | 5 |

1. Medals(MedalsId, MemberId, Medal):
	1. CK: [1] MedalsId, [2] MemberId, Medal.
	2. FK: [1] MemberId references Member(MemberId)
	3. A surrogate key, MedalsId, is created as the primary key.

Medal:

|  |  |  |
| --- | --- | --- |
| **Member\_Id (FK)** | **Medal** | **MedalId (optional: for simple PK)** |
| 123 | Zinc | 1 |
| 123 | Aluminium | 2 |
| 123 | Mercury | 3 |
| 222 |  |  |

All columns in the table above are not nullable.

***Example:***

A class Rectangle has three attributes:

* Length
* Width
* \area: derived.

What may the relational schema look like?

One solution:

Rectangle(RectangleId, Length, Width)

with a view Rect define as

select RectangleId, Length, Width, Length \* Width as Area
from Rectangle;

Alternatively, using virtual column:

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

**Associations**

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

1. Assume that R1\_Id is the primary key of the relation R1 (e.g. Member).
2. R1\_Id is a foreign key of Rm (e.g project) referencing R1(R1\_Id).
3. The name R1\_Id may need to be renamed.
4. R1\_Id is not null in Rm 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 you have composite or multi-valued attributes of the relationship, you should consider promoting the association to an association class or a class in your UML class diagram.

M1 is the owner of project P1

Member (1..1) owns (0..\*) projects.

Good:

Project:

|  |  |  |  |
| --- | --- | --- | --- |
| ProjectId | Name | … | OwnerId (FK references Member(MemberId); singled-valued |
| P1 |  |  | M1 |

Member: bad, no, no.

|  |  |  |  |
| --- | --- | --- | --- |
| MemberId | FName |  | OwnedProjectId (multi-values) |
| M1 | Bun |  | P1, P3, P9 (no, no) |
|  |  |  |  |

**A2.** (basic) For a many to many association between classes CA and CB, create a new relation RAB(RA\_Id, RB\_Id).

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

ProjMember (0..\*) relates to Roles (0..\*): cannot just use attributes.

PM1 relates roles R1, R2, R3
PM2 has the roles R1 and R4.

New table:

|  |  |  |
| --- | --- | --- |
| PM\_Id (FK) | RoleId (FK) |  |
| PM1 | R1 |  |
| PM1 | R2 |  |
| PM1 | R3 |  |
| PM2 | R1 |  |
| PM2 | R4 |  |

**A3.** 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.)

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

1. You should consider using binary associations instead.

**Example:**

Consider the classes Order and OrderItem which have a (1..1) to (0..\*) association. The association has an attribute 'packed'.

What may the relational schema look like?

OrderItem(OrderItemId, ..., OrderId, packed):

* OrderId is a foreign key referencing Order(OrderId)

**Example:**

Consider the class User and Account, which has a one to one association.

What may the relational schema look like?

**Example:**

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

What may the relational schema look like?

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).