# DASC 5333 Database Systems for Data Science

## 2/13/2024 (self - annotation)

**The Relational Model**

by K. Yue

**1. Introduction**

**Tips: be as specific as possible.**

**1.1 Data Model**

* There are many *data models* used by database systems.
* The **data models** of database systems define how data is organized, structured, connected, processed, and queried in the databases.
* It is important to recognize the basic data structures used by these models.
* Examples:
  1. Relational model: set-theoretic relation/table: unordered, no duplicate
     + {1,2} = {2,1} = {1,1,1,2,2}
  2. Excel: table, and to be more exact, a two dimensional array
     + Ordered: [1,2] <> [2,1]
     + Allow duplicates: [1,1,1,2] <> [1,2]
  3. Hierarchical model (old): tree
  4. Network model (old): graph
  5. Object-oriented model: directed graph
  6. XML: tree with many different types of nodes, plus sets of attributes.
  7. Cassandra: columnar or wild column model
  8. MongoDB: document model
  9. Neo4J: graph model

**1.2 The Relational Model: an introduction**

* The basic relational data model in layman terms:
  + A database is composed of a collection of *tables* (relations).
  + A table contains many *rows* (tuples) and *columns* (attributes/fields).
  + Each row contains many *column values*.
  + Every row of a table has the same set of columns.
  + Values of the same column have the same data *type*.
  + A *candidate key* of a table is a *minimal unique identifier* of a row in the table.
  + A *primary key* is a selected candidate key (for storing the table).
  + *Alternative/secondary keys* are candidate keys not selected as the primary key.
  + A *foreign key* of a relation refers to a primary key of another relation (known as the parent or refered table).

More theoretically:

* The (theoretical) relational model is based on the concept of a relation.
* It is a *set-theoretic* model: the definitions are based on mathematical sets.
* If you are note familar with set theory, read about it. This is a basic, short, good, and good-enough introduction: <https://www.ucl.ac.uk/~ucahmto/0005_2021/Ch2.S1.html> (note that in the set builder form, the author used ":" to represent "such that". We usually use "|" instead.)
* Note that practical DBMS do not implement the pure relational model.
* In the theoretical relational model:
  1. An *attribute* (*column/field*) is a name.
  2. A *domain* is a *set* of values an attribute can take.
     1. It is the set of values of the*data type* of the attribute.
     2. The value of an attribute should be *atomic* (cannot be divided into smaller components with individual meanings):
        1. If all attributes of a relation are atomic, the relation is said to be in *First Normal Form*.
     3. *Null* may or may not be an acceptable value for an attribute.
  3. A *relation schema*, R, is a *set* of attributes A1, A2,…,An with their domains D1, D2,…, Dn.

A screenshot of a computer

Description automatically generated

* 1. A *tuple* (*row*) is a *set* of *mapping* of a *set* of attributes to a *set* of values: Ai -> di where di ∈ (belongs to) Di, for i = 1 to n (∈: belongs to)

A screenshot of a computer

Description automatically generatedA *relation* (*instance*) is a set of tuples.

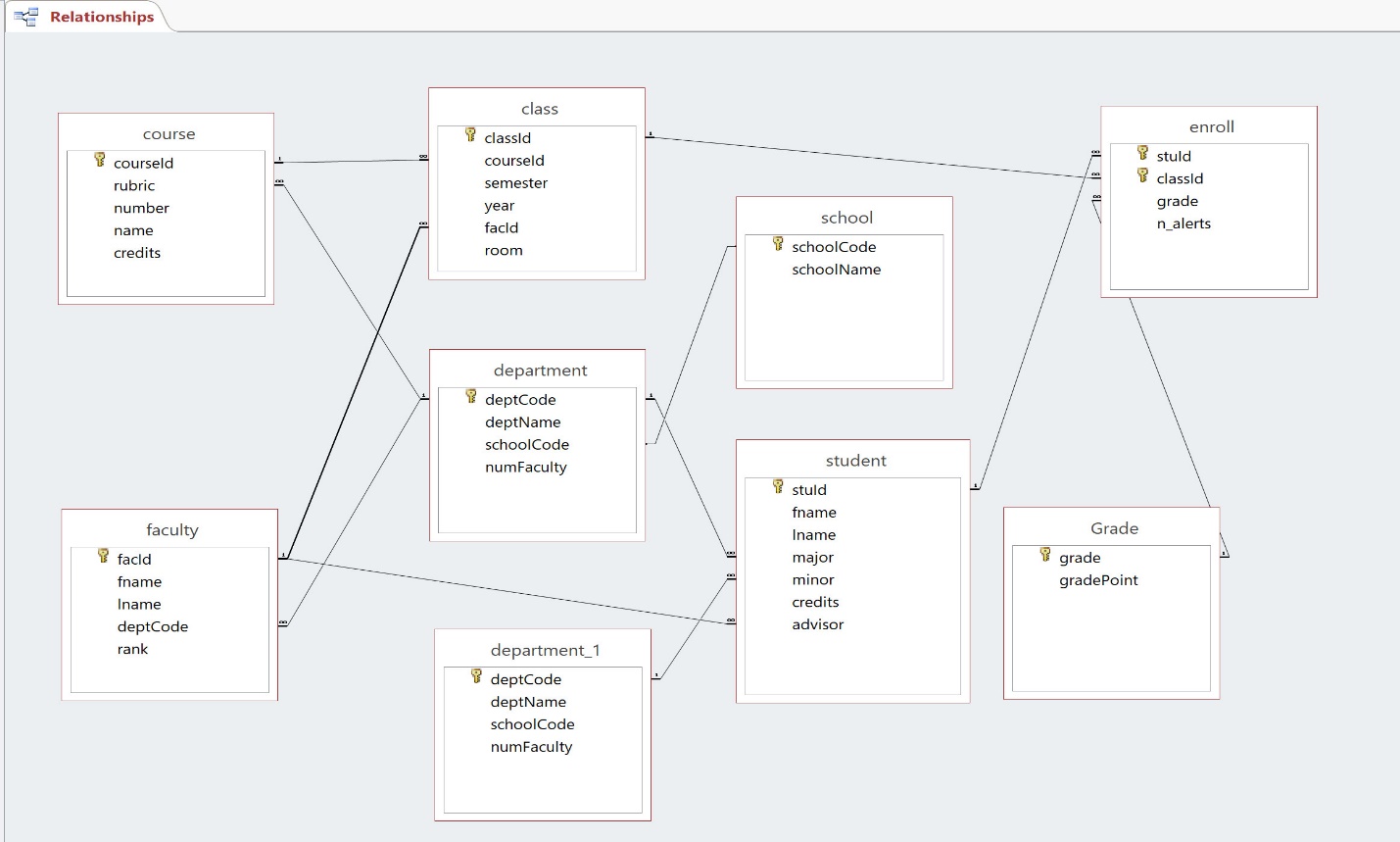
* 1. The *degree* (or *arity*) of a relation is the number of attributes in its schema.
* Some advantages of the *relational model* and relational DBMS (as compared to other databases)
  1. Strong mathematical foundation
  2. Simple
  3. Strong design theory
  4. Strong support of data integrity and consistency
  5. Strong support of transactions
  6. Strong industrial support and community
  7. High popularity
* Some disadvantages of the relational model and relational DBMS:
  1. The data model may not match the problem requirements well.
  2. Impedance mismatch with object-oriented models.
  3. Do not scale well.
  4. Structured data ay be too restrictive.

***Example:***

Toyu: A drastically simplified university: toyu

*Relationship* Diagram (in MS Access):

1. Each rectange is a relation, not a class.
2. Relationships are usually between foreign keys and refered primary keys.



A version of an Entity-Relationship Diagram (created by using MySQL Workbench).

1. This version of ER diagram has more commonality with Access relationship diagram.
2. A rectangle is a relation.
3. In other versions of ER diagrams, a rectangle is an entity (similar to a class).

A computer screen shot of a diagram

Description automatically generated

MS Access: [toyu.accdb](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/access/toyu.accdb)

Toyu in MySQL: [createtoyu.sql.txt](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2024/notes/query/createtoyu.sql.txt)

CREATE TABLE IF NOT EXISTS Class (  
    classId     INT NOT NULL,  
    courseId    INT NOT NULL,  
    semester    VARCHAR(10) NOT NULL,  
    year        DECIMAL(4,0) NOT NULL,  
    facId       INT NOT NULL,  
    room        VARCHAR(6) NULL,  
    CONSTRAINT Class\_classId\_pk PRIMARY KEY (classId),  
    CONSTRAINT Class\_courseId\_fk FOREIGN KEY (courseId)  
        REFERENCES Course(courseId) ON DELETE CASCADE,  
    CONSTRAINT Class\_facId\_fk FOREIGN KEY (facId)  
        REFERENCES Faculty (facId) ON DELETE CASCADE  
);

INSERT INTO Class(classId, courseId, semester, year, facId, room) VALUES  
    (10000,2000,'Fall',2019,1011,'D241'),  
    (10001,2001,'Fall',2019,1011,'D242'),  
    (10002,2002,'Fall',2019,1012,'D136'),  
    (10003,2020,'Fall',2019,1014,'D241'),  
    (10004,2021,'Fall',2019,1014,'D241'),  
    (10005,2040,'Fall',2019,1015,'D237'),  
    (10006,2041,'Fall',2019,1019,'D217'),  
    (10007,2060,'Fall',2019,1020,'B101'),  
    (10008,2080,'Fall',2019,1018,'D241'),  
    (11000,2000,'Spring',2020,1011,'D241'),  
    (11001,2001,'Spring',2020,1012,'D242'),  
    (11002,2002,'Spring',2020,1013,'D136'),  
    (11003,2020,'Spring',2020,1016,'D217'),  
    (11004,2061,'Spring',2020,1018,'B101');

MariaDB [toyu]> SELECT \* FROM class;  
+---------+----------+----------+------+-------+------+  
| classId | courseId | semester | year | facId | room |  
+---------+----------+----------+------+-------+------+  
|   10000 |     2000 | Fall     | 2019 |  1011 | D241 |  
|   10001 |     2001 | Fall     | 2019 |  1011 | D242 |  
|   10002 |     2002 | Fall     | 2019 |  1012 | D136 |  
|   10003 |     2020 | Fall     | 2019 |  1014 | D241 |  
|   10004 |     2021 | Fall     | 2019 |  1014 | D241 |  
|   10005 |     2040 | Fall     | 2019 |  1015 | D237 |  
|   10006 |     2041 | Fall     | 2019 |  1019 | D217 |  
|   10007 |     2060 | Fall     | 2019 |  1020 | B101 |  
|   10008 |     2080 | Fall     | 2019 |  1018 | D241 |  
|   11000 |     2000 | Spring   | 2020 |  1011 | D241 |  
|   11001 |     2001 | Spring   | 2020 |  1012 | D242 |  
|   11002 |     2002 | Spring   | 2020 |  1013 | D136 |  
|   11003 |     2020 | Spring   | 2020 |  1016 | D217 |  
|   11004 |     2061 | Spring   | 2020 |  1018 | B101 |  
+---------+----------+----------+------+-------+------+  
14 rows in set (0.005 sec)

A tuple/row: {classId: 10004, courseId: 2021, semester: 'Fall', year: 2019, facId: 1014, room: 'D241'}. Note that it is a set of mappings from attribute names to attribute values.

It can also be represented as:

{classId: 10004, facId: 1014, room: 'D241', semester: 'Fall', year: 2019, courseId: 2021}

or

{10004, 2021, 'Fall', 2019, 1014, 'D241'} if the attribute names are assumed.

or using a more computer science-style notation.

(10004, 2021, 'Fall', 2019, 1014, 'D241')

Identify some examples of the database terms in the class table above as much as possible.

* Some important properties of a relation:
  + There is no duplicate tuple.
    - Because a relation is a set.
    - Consequence: the relational model does not support 'object identity' directly.
  + The relational model is known to be '*value-oriented*':
    - A row is a set of attribute values.
    - Two rows with the same values of all attributes are the same row.
    - Cannot store two duplicate rows in a table.
  + The terms tables and relations are not exactly the same. "Table" is a more generic term.
* Tuples within a relation are unordered.
  + Changing the order of displaying the tuples does not change (the meaning of) the relation.
* Attributes within a relation schema are unordered.
  + Changing the order of the attributes within a relation schema does not change the information stored in the relation.

Example:

+---------+----------+----------+------+-------+------+  
| classId | courseId | semester | year | facId | room |  
+---------+----------+----------+------+-------+------+  
|   10000 |     2000 | Fall     | 2019 |  1011 | D241 |  
|   10001 |     2001 | Fall     | 2019 |  1011 | D242 |  
|   10002 |     2002 | Fall     | 2019 |  1012 | D136 |  
|   10003 |     2020 | Fall     | 2019 |  1014 | D241 |  
|   10004 |     2021 | Fall     | 2019 |  1014 | D241 |  
|   10005 |     2040 | Fall     | 2019 |  1015 | D237 |  
|   10006 |     2041 | Fall     | 2019 |  1019 | D217 |  
|   10007 |     2060 | Fall     | 2019 |  1020 | B101 |  
|   10008 |     2080 | Fall     | 2019 |  1018 | D241 |  
|   11000 |     2000 | Spring   | 2020 |  1011 | D241 |  
|   11001 |     2001 | Spring   | 2020 |  1013 | D242 |  
|   11002 |     2002 | Spring   | 2020 |  1013 | D136 |  
|   11003 |     2020 | Spring   | 2020 |  1016 | D217 |  
|   11004 |     2061 | Spring   | 2020 |  1018 | B101 |  
+---------+----------+----------+------+-------+------+

and

+---------+------+----------+----------+------+-------+  
| classId | year | semester | courseId | room | facId |  
+---------+------+----------+----------+------+-------+  
|   11004 | 2020 | Spring   |     2061 | B101 |  1018 |  
|   10007 | 2019 | Fall     |     2060 | B101 |  1020 |  
|   10002 | 2019 | Fall     |     2002 | D136 |  1012 |  
|   11002 | 2020 | Spring   |     2002 | D136 |  1013 |  
|   11003 | 2020 | Spring   |     2020 | D217 |  1016 |  
|   10006 | 2019 | Fall     |     2041 | D217 |  1019 |  
|   10005 | 2019 | Fall     |     2040 | D237 |  1015 |  
|   10000 | 2019 | Fall     |     2000 | D241 |  1011 |  
|   11000 | 2020 | Spring   |     2000 | D241 |  1011 |  
|   10003 | 2019 | Fall     |     2020 | D241 |  1014 |  
|   10004 | 2019 | Fall     |     2021 | D241 |  1014 |  
|   10008 | 2019 | Fall     |     2080 | D241 |  1018 |  
|   10001 | 2019 | Fall     |     2001 | D242 |  1011 |  
|   11001 | 2020 | Spring   |     2001 | D242 |  1013 |  
+---------+------+----------+----------+------+-------+  
  
  
store the same information.

The second table can be obtained in SQL by:

SELECT classId, year, semester, courseId, room, facId  
FROM class  
ORDER BY room, facId;

**2. Null values**

* Generally used for representing missing information.
* SQL DBSM provide a method to test whether a value is null or not (IS NULL and IS NOT NULL).

***Example:***

-- students with no advisor  
SELECT s.\*  
FROM student AS s  
WHERE s.advisor IS NULL; -- unary operator

SELECT s.\*  
FROM student AS s  
WHERE 1; -- all students

SELECT s.\*  
FROM student AS s  
WHERE s.advisor <> NULL; -- binary equality test => empty set  
  
-- Show all students with a declared minor.  
SELECT DISTINCT s.\*  
FROM student AS s  
WHERE s.minor IS NOT NULL;  
  
-- Show enrollment without a n\_alerts value.  
SELECT e.\*  
FROM enroll AS e  
WHERE e.n\_alerts IS NULL;  
  
Result:

MariaDB [toyu]> -- students with no advisor  
MariaDB [toyu]> SELECT s.\*  
    -> FROM student AS s  
    -> WHERE s.advisor IS NULL;  
+--------+-----------+-------+-------+-------+------+---------+  
| stuId  | fname     | lname | major | minor | ach  | advisor |  
+--------+-----------+-------+-------+-------+------+---------+  
| 100003 | Catherine | Lim   | ITEC  | CINF  |   20 |    NULL |  
| 100007 | Ben       | Zico  | NULL  | NULL  |   16 |    NULL |  
| 100008 | Bill      | Ching | ARTS  | NULL  |   90 |    NULL |  
+--------+-----------+-------+-------+-------+------+---------+  
3 rows in set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> -- Show all students with a declared minor.  
MariaDB [toyu]> SELECT DISTINCT s.\*  
    -> FROM student AS s  
    -> WHERE s.minor IS NOT NULL;  
+--------+-----------+---------+-------+-------+------+---------+  
| stuId  | fname     | lname   | major | minor | ach  | advisor |  
+--------+-----------+---------+-------+-------+------+---------+  
| 100000 | Tony      | Hawk    | CSCI  | CINF  |   40 |    1011 |  
| 100001 | Mary      | Hawk    | CSCI  | CINF  |   35 |    1011 |  
| 100002 | David     | Hawk    | CSCI  | ITEC  |   66 |    1012 |  
| 100003 | Catherine | Lim     | ITEC  | CINF  |   20 |    NULL |  
| 100005 | Linda     | Johnson | CINF  | ENGL  |   13 |    1015 |  
| 100006 | Lillian   | Johnson | CINF  | ITEC  |   18 |    1016 |  
| 100009 | Linda     | King    | ARTS  | CSCI  |  125 |    1018 |  
+--------+-----------+---------+-------+-------+------+---------+  
7 rows in set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> -- Show enrollment without a n\_alerts value.  
MariaDB [toyu]> SELECT e.\*  
    -> FROM enroll AS e  
    -> WHERE e.n\_alerts IS NULL;  
+--------+---------+-------+----------+  
| stuId  | classId | grade | n\_alerts |  
+--------+---------+-------+----------+  
| 100001 |   10000 | NULL  |     NULL |  
| 100005 |   10003 | NULL  |     NULL |  
| 100004 |   10004 | B+    |     NULL |  
| 100006 |   10004 | C+    |     NULL |  
| 100006 |   10005 | A     |     NULL |  
| 100005 |   10006 | B+    |     NULL |  
+--------+---------+-------+----------+  
6 rows in set (0.000 sec)

**Null and Boolean Expressions**

1. MySQL does not have a Boolean data type. A Boolean value is converted to TINYINT: 0 as FALSE, otherwise TRUE.
2. If a Boolean value is expected, null is implicitly type converted to FALSE.
3. However, NULL is a special value different with 0 or empty string.
4. Comparing null to other values return false.

***Example:***

-- 1. Boolean values are TINYINT. FALSE is 0.  
SELECT FALSE,  
   TRUE; -- 0 and 1  
    
SELECT \*  
FROM student  
WHERE 0; -- empty set  
  
SELECT \*  
FROM student  
WHERE 1; -- all students  
  
SELECT \*  
FROM student  
WHERE 2697; -- all students  
  
SELECT \*  
FROM student  
WHERE '0';  
  
SELECT \*  
FROM student  
WHERE '145'; -- implicit data conversion. ‘145’ -> 145: true;

-- be mindful of implicit data type conversion.

-- tips: be explicit.  
  
-- warning: '' cannot be converted to a number.  
-- "Warning 1292 Truncated incorrect DOUBLE value: ''"  
SELECT \*  
FROM student  
WHERE '';  
  
-- warning: '' cannot be converted to a number.  
-- "Warning 1292 Truncated incorrect DOUBLE value: ''"  
SELECT \*  
FROM student  
WHERE 'Hello world';  
  
SELECT \*  
FROM student  
WHERE 1.49;

Result:

MariaDB [toyu]> SELECT FALSE,  
    -> TRUE;  
+-------+------+  
| FALSE | TRUE |  
+-------+------+  
|     0 |    1 |  
+-------+------+  
1 row in set (0.001 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> SELECT \*  
    -> FROM student  
    -> WHERE 0;  
Empty set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> SELECT \*  
    -> FROM student  
    -> WHERE 1;  
+--------+-----------+----------+-------+-------+------+---------+  
| stuId  | fname     | lname    | major | minor | ach  | advisor |  
+--------+-----------+----------+-------+-------+------+---------+  
| 100000 | Tony      | Hawk     | CSCI  | CINF  |   40 |    1011 |  
| 100001 | Mary      | Hawk     | CSCI  | CINF  |   35 |    1011 |  
| 100002 | David     | Hawk     | CSCI  | ITEC  |   66 |    1012 |  
| 100003 | Catherine | Lim      | ITEC  | CINF  |   20 |    NULL |  
| 100004 | Larry     | Johnson  | ITEC  | NULL  |   66 |    1017 |  
| 100005 | Linda     | Johnson  | CINF  | ENGL  |   13 |    1015 |  
| 100006 | Lillian   | Johnson  | CINF  | ITEC  |   18 |    1016 |  
| 100007 | Ben       | Zico     | NULL  | NULL  |   16 |    NULL |  
| 100008 | Bill      | Ching    | ARTS  | NULL  |   90 |    NULL |  
| 100009 | Linda     | King     | ARTS  | CSCI  |  125 |    1018 |  
| 100111 | Cathy     | Johanson | NULL  | NULL  |    0 |    1018 |  
+--------+-----------+----------+-------+-------+------+---------+  
11 rows in set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> SELECT \*  
    -> FROM student  
    -> WHERE 2697;  
+--------+-----------+----------+-------+-------+------+---------+  
| stuId  | fname     | lname    | major | minor | ach  | advisor |  
+--------+-----------+----------+-------+-------+------+---------+  
| 100000 | Tony      | Hawk     | CSCI  | CINF  |   40 |    1011 |  
| 100001 | Mary      | Hawk     | CSCI  | CINF  |   35 |    1011 |  
| 100002 | David     | Hawk     | CSCI  | ITEC  |   66 |    1012 |  
| 100003 | Catherine | Lim      | ITEC  | CINF  |   20 |    NULL |  
| 100004 | Larry     | Johnson  | ITEC  | NULL  |   66 |    1017 |  
| 100005 | Linda     | Johnson  | CINF  | ENGL  |   13 |    1015 |  
| 100006 | Lillian   | Johnson  | CINF  | ITEC  |   18 |    1016 |  
| 100007 | Ben       | Zico     | NULL  | NULL  |   16 |    NULL |  
| 100008 | Bill      | Ching    | ARTS  | NULL  |   90 |    NULL |  
| 100009 | Linda     | King     | ARTS  | CSCI  |  125 |    1018 |  
| 100111 | Cathy     | Johanson | NULL  | NULL  |    0 |    1018 |  
+--------+-----------+----------+-------+-------+------+---------+  
11 rows in set (0.001 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> SELECT \*  
    -> FROM student  
    -> WHERE '0';  
Empty set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> SELECT \*  
    -> FROM student  
    -> WHERE '145';  
+--------+-----------+----------+-------+-------+------+---------+  
| stuId  | fname     | lname    | major | minor | ach  | advisor |  
+--------+-----------+----------+-------+-------+------+---------+  
| 100000 | Tony      | Hawk     | CSCI  | CINF  |   40 |    1011 |  
| 100001 | Mary      | Hawk     | CSCI  | CINF  |   35 |    1011 |  
| 100002 | David     | Hawk     | CSCI  | ITEC  |   66 |    1012 |  
| 100003 | Catherine | Lim      | ITEC  | CINF  |   20 |    NULL |  
| 100004 | Larry     | Johnson  | ITEC  | NULL  |   66 |    1017 |  
| 100005 | Linda     | Johnson  | CINF  | ENGL  |   13 |    1015 |  
| 100006 | Lillian   | Johnson  | CINF  | ITEC  |   18 |    1016 |  
| 100007 | Ben       | Zico     | NULL  | NULL  |   16 |    NULL |  
| 100008 | Bill      | Ching    | ARTS  | NULL  |   90 |    NULL |  
| 100009 | Linda     | King     | ARTS  | CSCI  |  125 |    1018 |  
| 100111 | Cathy     | Johanson | NULL  | NULL  |    0 |    1018 |  
+--------+-----------+----------+-------+-------+------+---------+  
11 rows in set (0.001 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> -- warning: '' cannot be converted to a number.  
MariaDB [toyu]> -- "Warning 1292 Truncated incorrect DOUBLE value: ''"  
MariaDB [toyu]> SELECT \*  
    -> FROM student  
    -> WHERE '';  
Empty set, 1 warning (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> -- warning: '' cannot be converted to a number.  
MariaDB [toyu]> -- "Warning 1292 Truncated incorrect DOUBLE value: ''"  
MariaDB [toyu]> SELECT \*  
    -> FROM student  
    -> WHERE 'Hello world';  
Empty set, 1 warning (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> SELECT \*  
    -> FROM student  
    -> WHERE 1.49;  
+--------+-----------+----------+-------+-------+------+---------+  
| stuId  | fname     | lname    | major | minor | ach  | advisor |  
+--------+-----------+----------+-------+-------+------+---------+  
| 100000 | Tony      | Hawk     | CSCI  | CINF  |   40 |    1011 |  
| 100001 | Mary      | Hawk     | CSCI  | CINF  |   35 |    1011 |  
| 100002 | David     | Hawk     | CSCI  | ITEC  |   66 |    1012 |  
| 100003 | Catherine | Lim      | ITEC  | CINF  |   20 |    NULL |  
| 100004 | Larry     | Johnson  | ITEC  | NULL  |   66 |    1017 |  
| 100005 | Linda     | Johnson  | CINF  | ENGL  |   13 |    1015 |  
| 100006 | Lillian   | Johnson  | CINF  | ITEC  |   18 |    1016 |  
| 100007 | Ben       | Zico     | NULL  | NULL  |   16 |    NULL |  
| 100008 | Bill      | Ching    | ARTS  | NULL  |   90 |    NULL |  
| 100009 | Linda     | King     | ARTS  | CSCI  |  125 |    1018 |  
| 100111 | Cathy     | Johanson | NULL  | NULL  |    0 |    1018 |  
+--------+-----------+----------+-------+-------+------+---------+  
11 rows in set (0.000 sec)

***Example:***

-- 2. If a Boolean value is expected, null is implicitly type-converted to FALSE.  
SELECT e.\*  
FROM enroll AS e  
WHERE e.n\_alerts;

Result:

MariaDB [toyu]> SELECT e.\*  
    -> FROM enroll AS e  
    -> WHERE e.n\_alerts;  
+--------+---------+-------+----------+  
| stuId  | classId | grade | n\_alerts |  
+--------+---------+-------+----------+  
| 100002 |   10000 | B-    |        3 |  
| 100000 |   10001 | A     |        2 |  
| 100000 |   10002 | B+    |        1 |  
| 100002 |   10002 | B+    |        2 |  
| 100002 |   10003 | D     |        4 |  
| 100000 |   10004 | A-    |        1 |  
| 100007 |   10007 | F     |        4 |  
| 100000 |   11001 | D     |        4 |  
+--------+---------+-------+----------+  
8 rows in set (0.000 sec)

***Example:***

-- 3. null is a special value different with 0 or empty string.  
SELECT FALSE IS NULL,    
   TRUE IS NULL,  
   0 IS NULL,  
   1 IS NULL,  
   "" IS NULL,  
   "Hey" IS NULL,  
   NULL IS NULL,  
   NULL IS NOT NULL;

Result:

MariaDB [toyu]> SELECT FALSE IS NULL,  
    -> TRUE IS NULL,  
    -> 0 IS NULL,  
    -> 1 IS NULL,  
    -> "" IS NULL,  
    -> "Hey" IS NULL,  
    -> NULL IS NULL,  
    -> NULL IS NOT NULL;  
+---------------+--------------+-----------+-----------+------------+---------------+--------------+------------------+  
| FALSE IS NULL | TRUE IS NULL | 0 IS NULL | 1 IS NULL | "" IS NULL | "Hey" IS NULL | NULL IS NULL | NULL IS NOT NULL |  
+---------------+--------------+-----------+-----------+------------+---------------+--------------+------------------+  
|             0 |            0 |         0 |         0 |          0 |             0 |            1 |                0 |  
+---------------+--------------+-----------+-----------+------------+---------------+--------------+------------------+  
1 row in set (0.000 sec)

***Example:***

-- 4. Comparing null to other values return null, which is converted to false.  
SELECT NULL > 3,  
   NULL <= 3,  
   5 >= NULL,  
   5 < NULL,  
   NULL > NULL,  
   NULL <= NULL;  
    
SELECT \*  
FROM student  
WHERE NULL > 3;  
  
-- Comparisons must be mindful of null.  
SELECT e.\*  
FROM enroll AS e  
WHERE e.n\_alerts >= 2;  
  
SELECT e.\*  
FROM enroll AS e  
WHERE e.n\_alerts < 2;  
  
SELECT e.\*  
FROM enroll AS e;  
  
-- Q. List all enrollment records without 2 or more n\_alerts.  
-- Naive solution  
SELECT e.\*  
FROM enroll AS e  
WHERE e.n\_alerts < 2;  
  
-- Q. List all enrollment records without 2 or more n\_alerts.  
-- More likely solution  
SELECT e.\*  
FROM enroll AS e  
WHERE e.n\_alerts IS NULL  
OR e.n\_alerts < 2;  
  
-- Q. List all enrollment records without a value in n\_alerts.  
-- incorrect answer.  
SELECT e.\*  
FROM enroll AS e  
WHERE e.n\_alerts <> NULL;  
  
-- Q. List all enrollment records without a value in n\_alerts.  
-- correct answer.  
SELECT e.\*  
FROM enroll AS e  
WHERE e.n\_alerts IS NOT NULL;

Result:

MariaDB [toyu]> -- 4. Comparing null to other values return null, which is converted to false.  
MariaDB [toyu]> SELECT NULL > 3,  
    -> NULL <= 3,  
    -> 5 >= NULL,  
    -> 5 < NULL,  
    -> NULL > NULL,  
    -> NULL <= NULL;  
+----------+-----------+-----------+----------+-------------+--------------+  
| NULL > 3 | NULL <= 3 | 5 >= NULL | 5 < NULL | NULL > NULL | NULL <= NULL |  
+----------+-----------+-----------+----------+-------------+--------------+  
|     NULL |      NULL |      NULL |     NULL |        NULL |         NULL |  
+----------+-----------+-----------+----------+-------------+--------------+  
1 row in set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> SELECT \*  
    -> FROM student  
    -> WHERE NULL > 3;  
Empty set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> -- Comparisons must be mindful of null.  
MariaDB [toyu]> SELECT e.\*  
    -> FROM enroll AS e  
    -> WHERE e.n\_alerts >= 2;  
+--------+---------+-------+----------+  
| stuId  | classId | grade | n\_alerts |  
+--------+---------+-------+----------+  
| 100002 |   10000 | B-    |        3 |  
| 100000 |   10001 | A     |        2 |  
| 100002 |   10002 | B+    |        2 |  
| 100002 |   10003 | D     |        4 |  
| 100007 |   10007 | F     |        4 |  
| 100000 |   11001 | D     |        4 |  
+--------+---------+-------+----------+  
6 rows in set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> SELECT e.\*  
    -> FROM enroll AS e  
    -> WHERE e.n\_alerts < 2;  
+--------+---------+-------+----------+  
| stuId  | classId | grade | n\_alerts |  
+--------+---------+-------+----------+  
| 100000 |   10000 | A     |        0 |  
| 100001 |   10001 | A-    |        0 |  
| 100000 |   10002 | B+    |        1 |  
| 100000 |   10003 | C     |        0 |  
| 100004 |   10003 | A     |        0 |  
| 100000 |   10004 | A-    |        1 |  
| 100005 |   10004 | A-    |        0 |  
| 100005 |   10005 | A-    |        0 |  
| 100008 |   10007 | C-    |        0 |  
| 100007 |   10008 | A-    |        0 |  
+--------+---------+-------+----------+  
10 rows in set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> SELECT e.\*  
    -> FROM enroll AS e;  
+--------+---------+-------+----------+  
| stuId  | classId | grade | n\_alerts |  
+--------+---------+-------+----------+  
| 100000 |   10000 | A     |        0 |  
| 100001 |   10000 | NULL  |     NULL |  
| 100002 |   10000 | B-    |        3 |  
| 100000 |   10001 | A     |        2 |  
| 100001 |   10001 | A-    |        0 |  
| 100000 |   10002 | B+    |        1 |  
| 100002 |   10002 | B+    |        2 |  
| 100000 |   10003 | C     |        0 |  
| 100002 |   10003 | D     |        4 |  
| 100004 |   10003 | A     |        0 |  
| 100005 |   10003 | NULL  |     NULL |  
| 100000 |   10004 | A-    |        1 |  
| 100004 |   10004 | B+    |     NULL |  
| 100005 |   10004 | A-    |        0 |  
| 100006 |   10004 | C+    |     NULL |  
| 100005 |   10005 | A-    |        0 |  
| 100006 |   10005 | A     |     NULL |  
| 100005 |   10006 | B+    |     NULL |  
| 100007 |   10007 | F     |        4 |  
| 100008 |   10007 | C-    |        0 |  
| 100007 |   10008 | A-    |        0 |  
| 100000 |   11001 | D     |        4 |  
+--------+---------+-------+----------+  
22 rows in set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> -- Q. List all enrollment records without 2 or more n\_alerts.  
MariaDB [toyu]> -- Naive solution  
MariaDB [toyu]> SELECT e.\*  
    -> FROM enroll AS e  
    -> WHERE e.n\_alerts < 2;  
+--------+---------+-------+----------+  
| stuId  | classId | grade | n\_alerts |  
+--------+---------+-------+----------+  
| 100000 |   10000 | A     |        0 |  
| 100001 |   10001 | A-    |        0 |  
| 100000 |   10002 | B+    |        1 |  
| 100000 |   10003 | C     |        0 |  
| 100004 |   10003 | A     |        0 |  
| 100000 |   10004 | A-    |        1 |  
| 100005 |   10004 | A-    |        0 |  
| 100005 |   10005 | A-    |        0 |  
| 100008 |   10007 | C-    |        0 |  
| 100007 |   10008 | A-    |        0 |  
+--------+---------+-------+----------+  
10 rows in set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> -- Q. List all enrollment records without 2 or more n\_alerts.  
MariaDB [toyu]> -- More likely solution  
MariaDB [toyu]> SELECT e.\*  
    -> FROM enroll AS e  
    -> WHERE e.n\_alerts IS NULL  
    -> OR e.n\_alerts < 2;  
+--------+---------+-------+----------+  
| stuId  | classId | grade | n\_alerts |  
+--------+---------+-------+----------+  
| 100000 |   10000 | A     |        0 |  
| 100001 |   10000 | NULL  |     NULL |  
| 100001 |   10001 | A-    |        0 |  
| 100000 |   10002 | B+    |        1 |  
| 100000 |   10003 | C     |        0 |  
| 100004 |   10003 | A     |        0 |  
| 100005 |   10003 | NULL  |     NULL |  
| 100000 |   10004 | A-    |        1 |  
| 100004 |   10004 | B+    |     NULL |  
| 100005 |   10004 | A-    |        0 |  
| 100006 |   10004 | C+    |     NULL |  
| 100005 |   10005 | A-    |        0 |  
| 100006 |   10005 | A     |     NULL |  
| 100005 |   10006 | B+    |     NULL |  
| 100008 |   10007 | C-    |        0 |  
| 100007 |   10008 | A-    |        0 |  
+--------+---------+-------+----------+  
16 rows in set (0.000 sec) MariaDB [toyu]> -- Q. List all enrollment records without a value in n\_alerts.  
  
MariaDB [toyu]> -- incorrect answer.  
MariaDB [toyu]> SELECT e.\*  
    -> FROM enroll AS e  
    -> WHERE e.n\_alerts <> NULL;  
Empty set (0.000 sec)  
  
MariaDB [toyu]>  
MariaDB [toyu]> -- Q. List all enrollment records without a value in n\_alerts.  
MariaDB [toyu]> -- correct answer.  
MariaDB [toyu]> SELECT e.\*  
    -> FROM enroll AS e  
    -> WHERE e.n\_alerts IS NOT NULL;  
+--------+---------+-------+----------+  
| stuId  | classId | grade | n\_alerts |  
+--------+---------+-------+----------+  
| 100000 |   10000 | A     |        0 |  
| 100002 |   10000 | B-    |        3 |  
| 100000 |   10001 | A     |        2 |  
| 100001 |   10001 | A-    |        0 |  
| 100000 |   10002 | B+    |        1 |  
| 100002 |   10002 | B+    |        2 |  
| 100000 |   10003 | C     |        0 |  
| 100002 |   10003 | D     |        4 |  
| 100004 |   10003 | A     |        0 |  
| 100000 |   10004 | A-    |        1 |  
| 100005 |   10004 | A-    |        0 |  
| 100005 |   10005 | A-    |        0 |  
| 100007 |   10007 | F     |        4 |  
| 100008 |   10007 | C-    |        0 |  
| 100007 |   10008 | A-    |        0 |  
| 100000 |   11001 | D     |        4 |  
+--------+---------+-------+----------+  
16 rows in set (0.000 sec)

**Interpretation of null values**

* Three possible interpretations:
  1. Not applicable.
  2. Missing value.
  3. No information at all.

***Example:***

Consider the attribute SpouseName. A Null value may mean:

1. not applicable: the person is not married.
2. missing information: the person is married but we do not have the name of the spouse.
3. no information at all: we do not know whether the person is married or not.

How do we distinguish between the three meanings of the null value in this case?

By using an extra attribute, such as MaritalStatus.

|  |  |  |  |
| --- | --- | --- | --- |
| **...** | **SpouseName** | **MaritalStatus** | **...** |
|  | Null | Married |  |
|  | Null | Not married |  |
|  | Null | Null |  |

**3. Keys**

* A*set* of attributes K is a *candidate key* (CK) of a relation R if it minimally identifies a tuple at any time:
  1. Uniqueness: No two tuples of R have the same value of K.
  2. Minimality: No proper subset of K has the uniqueness property.

Enrol table: CK: {stuId, classId}

Department table:

{deptCode, schoolCOde} not a CK: unique not minimal.prin

* A candidate key is a property of the semantic (meaning) of a relation.
* In other words, a candidate key is the result of the requirements of an application.
* A relation always has at least one candidate keys. Why?
  1. Because a relation instance r is a set of rows, no two rows will have the exact same values.
  2. Thus, the relation schema R by itself satisfy the uniqueness properties.
  3. R or one of its proper subsets will satisfy the minimal properties as extraneous attributes are removed.
* A set of attributes may be a candidate key for a relation R but not a candidate key of another relation S.
* For *any* relation instance, the candidate key of a tuple must have an unique value.
* Whether a set of attributes is a key or not depend on the semantic (meaning) of the relation on modeling the application.
* A *primary* key is a selected candidate key for a relation in the DBMS. It is used for practical purpose (of how the relation is stored physically) and does not have a special meaning in the theory of the relational model.
* An*alternate/secondary* key is a candidate key that is not the primary key.
* Questions people may ask:
  1. What is the key of the database toyu? Wrong question.
  2. What is the primary key of the database toyu? more precise but still wrong question.
  3. What is the key of the school table of the toyu database? good question but a bit ambiguous. Answer: schoolCode.
  4. What is the primary key of the school table of the toyu database? schoolCode.
  5. What are the candidate keys of the school table of the toyu database? [1] schoolCode, [2] schoolName.
  6. What are the secondary/alternative keys of the school table of the toyu database? schoolName.

***Example:***

Consider:  
MariaDB [toyu]> SELECT \* FROM faculty;  
+-------+----------+----------+----------+---------------------+  
| facId | fname    | lname    | deptCode | rank                |  
+-------+----------+----------+----------+---------------------+  
|  1011 | Paul     | Smith    | CSCI     | Professor           |  
|  1012 | Mary     | Tran     | CSCI     | Associate Professor |  
|  1013 | David    | Love     | CSCI     | NULL                |  
|  1014 | Sharon   | Mannes   | CSCI     | Assistant Professor |  
|  1015 | Daniel   | Kim      | CINF     | Professor           |  
|  1016 | Andrew   | Byre     | CINF     | Associate Professor |  
|  1017 | Deborah  | Gump     | ITEC     | Professor           |  
|  1018 | Art      | Allister | ARTS     | Assistant Professor |  
|  1019 | Benjamin | Yu       | ITEC     | Lecturer            |  
|  1020 | Katrina  | Bajaj    | ENGL     | Lecturer            |  
|  1021 | Jorginlo | Neymar   | ACCT     | Assistant Professor |  
+-------+----------+----------+----------+---------------------+  
11 rows in set (0.001 sec)  
  
MariaDB [toyu]> SELECT \* FROM department;  
+----------+------------------------------+------------+----------+  
| deptCode | deptName                     | schoolCode | numStaff |  
+----------+------------------------------+------------+----------+  
| ACCT     | Accounting                   | BUS        |       10 |  
| ARTS     | Arts                         | HSH        |        5 |  
| CINF     | Computer Information Systems | CSE        |        5 |  
| CSCI     | Computer Science             | CSE        |       12 |  
| ENGL     | English                      | HSH        |       12 |  
| ITEC     | Information Technology       | CSE        |        4 |  
| MATH     | Mathematics                  | CSE        |        7 |  
+----------+------------------------------+------------+----------+  
7 rows in set (0.001 sec)

* deptCode is a candidate key of the relation department. Assumptions made:
  1. Each tuple in the department relation represents an unique department.
  2. Each department has an unique code, or no two departments can have the same code.
* deptName is also a candidate key of the relation department. Assumptions made:
  1. Each tuple in the department relation represents an unique department.
  2. Each department has an unique name, or no two departments can have the same name.
* deptCode is not a candidate key of the relation faculty. Assumptions made:
  1. Each tuple in the faculty relation represents a faculty member.
  2. Each faculty serves only one department.
  3. A department may have many faculty members.

Consider the tables student and enroll:

* stuid is a candidate key of the relation student. Assumptions made:
  1. Each tuple in the student relation represents an unique student.
  2. Each student has an unique student id, or no two students can have the same stuid.
* stuid is not a candidate key of the relation enroll. Assumptions made:
  1. Each tuple in the Enroll relation represents the enrollment of a student in a class, ending up with a grade.
  2. A student can take many classes.
* Note again that a candidate key is a property of a relation.

***Classroom Exercise:***

(1) Give a realistic example of a relation with two candidate keys. State the assumptions you have made.

(2) A relation R has an *arity* of 4. What are the possible minimum and maximum number of superkeys of R? What are the possible minimum and maximum number of candidate keys of R?

What about the general case?

**More about keys:**

* A key is *simple* if it has only one attribute.
* A key is a *composite* key if it has more than one attributes.
* A key is a *compound* key if it is a composite and each attribute in the key is a foreign key.
* Every relation has at least one candidate key.
* An attribute that appears in one or more candidate keys is a *prime attribute* (or key attribute).

Department:

CK: (1) {deptCode}, (2) {deptName}

Prime (key) attributes: deptCode, deptName  
non-prime attributes: numStaff, schoolCode

* An attribute that does not appears in any candidate key is a *non-prime* (non-key) attribute.
* A superkey of a relation is a set of attributes that uniquely identify a row (uniqueness). It may not be minimal. (A Superkey may have extraneous attributes not needed for unique identification.)
* A *foreign key* of a relation is a set of attributes that is a candidate key in another relation. The other relation is sometimes called the parent (or referenced) table of the foreign key.
* A foreign key may or may not have null value. It depends on the problem requirements.

Consider the example above. The attribute deptCode is a foreign key in the relation faculty, referencing department(deptCode)

* A deptCode in the department relation must be referring to a department in the database, identified by deptCode in the relation department (referential integrity).
* In Relational DB, rows (data) from different tables are linked together using *foreign keys*.
* A foreign key of a child table links to a primary key of the parent table.

Likewise, stuid is a foreign key in the relation Enroll: stuid references student(stuid)

Note that the student table has three foreign keys.

***Example:***

The list of all foreign keys in toyu:

1. Student(advisor) references Faculty(facId)  
2. Student(major) references Department(deptCode)  
3. Student(minor) references Department(deptCode)  
4. Faculty(deptCode) references Department(deptCode)  
5. Department(schoolCode) references School(schoolCode)  
6. Enroll(stuId) references Student(stuId)  
7. Enroll(classId) references Class(classId)  
8. Enroll(grade) references Grade(grade)  
9. Class(courseId) references Course(courseId)  
10. Class(facId) references Faculty(facId)

* The INNER JOIN operation can be used to join tables through foreign keys.

***Example:***

-- INNER JOIN  
SELECT \* FROM faculty;  
SELECT \* FROM department;  
SELECT s.fname, s.lname, s.advisor, f.`rank`  
FROM student AS s INNER JOIN faculty AS f  
   ON (s.advisor = f.facId);  
    
Result:

MariaDB [toyu]> SELECT \* FROM faculty;  
+-------+----------+----------+----------+---------------------+  
| facId | fname    | lname    | deptCode | rank                |  
+-------+----------+----------+----------+---------------------+  
|  1011 | Paul     | Smith    | CSCI     | Professor           |  
|  1012 | Mary     | Tran     | CSCI     | Associate Professor |  
|  1013 | David    | Love     | CSCI     | NULL                |  
|  1014 | Sharon   | Mannes   | CSCI     | Assistant Professor |  
|  1015 | Daniel   | Kim      | CINF     | Professor           |  
|  1016 | Andrew   | Byre     | CINF     | Associate Professor |  
|  1017 | Deborah  | Gump     | ITEC     | Professor           |  
|  1018 | Art      | Allister | ARTS     | Assistant Professor |  
|  1019 | Benjamin | Yu       | ITEC     | Lecturer            |  
|  1020 | Katrina  | Bajaj    | ENGL     | Lecturer            |  
|  1021 | Jorginlo | Neymar   | ACCT     | Assistant Professor |  
+-------+----------+----------+----------+---------------------+  
11 rows in set (0.000 sec)  
  
MariaDB [toyu]> SELECT \* FROM department;  
+----------+------------------------------+------------+----------+  
| deptCode | deptName                     | schoolCode | numStaff |  
+----------+------------------------------+------------+----------+  
| ACCT     | Accounting                   | BUS        |       10 |  
| ARTS     | Arts                         | HSH        |        5 |  
| CINF     | Computer Information Systems | CSE        |        5 |  
| CSCI     | Computer Science             | CSE        |       12 |  
| ENGL     | English                      | HSH        |       12 |  
| ITEC     | Information Technology       | CSE        |        4 |  
| MATH     | Mathematics                  | CSE        |        7 |  
+----------+------------------------------+------------+----------+  
7 rows in set (0.000 sec)  
  
MariaDB [toyu]> SELECT s.fname, s.lname, s.advisor, f.`rank`  
    -> FROM student AS s INNER JOIN faculty AS f  
    -> ON (s.advisor = f.facId);  
+---------+----------+---------+---------------------+  
| fname   | lname    | advisor | rank                |  
+---------+----------+---------+---------------------+  
| Tony    | Hawk     |    1011 | Professor           |  
| Mary    | Hawk     |    1011 | Professor           |  
| David   | Hawk     |    1012 | Associate Professor |  
| Larry   | Johnson  |    1017 | Professor           |  
| Linda   | Johnson  |    1015 | Professor           |  
| Lillian | Johnson  |    1016 | Associate Professor |  
| Linda   | King     |    1018 | Assistant Professor |  
| Cathy   | Johanson |    1018 | Assistant Professor |  
+---------+----------+---------+---------------------+  
8 rows in set (0.000 sec)

**Introduction to UML**

by K. Yue

**1. Introduction to UML**

* UML: A set of graphical notations for object-oriented modeling.
* Wikipedia: "The Unified Modeling Language (UML) offers a way to visualize a system's architectural blueprints in a diagram."
* A standard maintained by OMG: [OMG's UML page](http://www.uml.org/#UML2.0https://www.uml.org/index.htmhttp://www.uml.org/index.htm).
* Two major versions:
  + Version 1.4.2: international standard released in 2005.
  + Version 2.5.1: released in 2017, added nested classifiers and improved behavior models. Specification: <https://www.omg.org/spec/UML>
* Two main types of diagrams:
  + Structure diagrams: model static structures.
  + Behavior diagrams:model dynamic behaviors.
* Version 2.5 has 15 diagrams: 7 structure diagrams and 8 behavior diagrams.
* Some Resources:
  + [OMG UML Resource](https://www.uml.org/resource-hub.htm)
  + [SPARX UML Tutorial](http://www.sparxsystems.com/uml-tutorial.html).
* We will focus on the *class diagram* only.

Class Diagram of UML 2.2 diagram (from Wikipedia):



**2. Class Diagrams (Emphasis on DB applications)**

**2.1 Introduction:**

* A *static* structure diagram in UML.
* "Describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes." -- from Wikipedia.
* Read "class diagram" from Wikipedia: <http://en.wikipedia.org/wiki/Class_diagram>.
* For a significantly better introduction by IBM: <http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/>.
* Two kinds of tools for drawing UML diagrams:
  + Graphical tools: main purposes are drawing diagrams (e.g., MS Visio, draw io, etc.)
  + Computer-Aided Software Engineering (CASE) tools: for software development with some understanding of the semantics of diagram elements (e.g., MagicDraw, IBM Rational Rhapsody, Visual Paradigm, Astah, etc.)
* We use Astah UML Editor
  + We will use *community version* in classroom demonstration, which is now deprecated.
  + Students can use the more powerful *student version* for free: search "astah student license".
* One may also use UML object diagrams to show objects and their associations of a snapshot of the system.

**2.2 A Simple Conceptual Modeling Process**

1. Examine application requirements to gain a good understanding of the problem.
2. Conduct an analysis to extract concepts that may have data requirements.
3. For each concept, design how should it be modeled? Major options are:
   1. by attributes
   2. by a class
   3. by associations between classes (including special associations, such as composition, aggregation, generalization, etc.)
   4. no need to model (as it does not represent data requirement)

These steps are repeated until the model reaches the necessary fidelity, accuracy, and precision.

***Example:***

**Problem**. A used car dealership application's subsystem: information about cars and their manufacturers.

**Specification description**: A car manufacturer has an unique id and name. A car maker may make many cars. For example, Honda, which may have an manufacturer id of 10001, makes Civic and Accord.

**Analysis and Design**

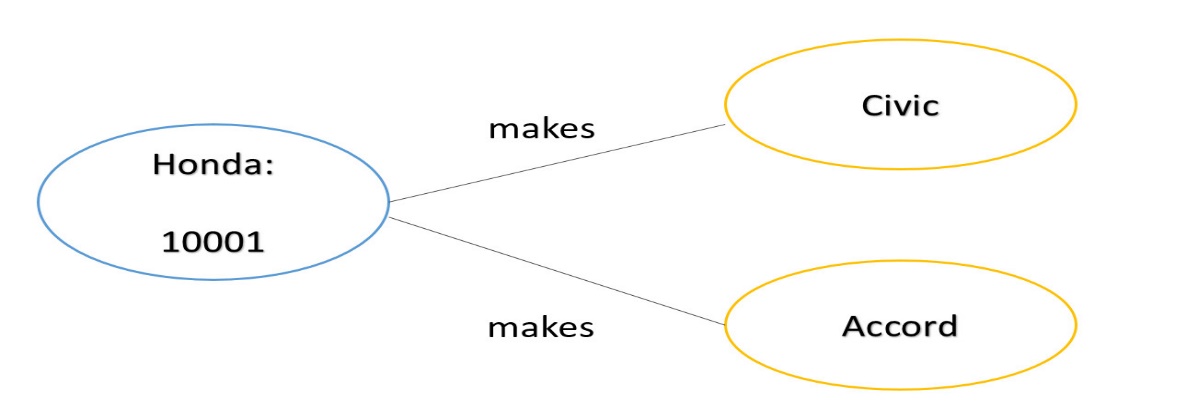
Some observations: identify concepts, ask questions.

1. Manufacturer: a class (template) that can be used to initiate many manufacturer objects (instances).
   1. Class/object
   2. Attributes
   3. Association between object
   4. No need to model (no data requirements)
2. Honda: an object of the class Manufacturer.
3. Resolve ambiguous terms: e.g., the term "manufacturer" may refer to the manufacturer class, or a particular manufacturer (a manufacturer object such as Honda).
4. Define synonym: manufacturer, car manufacturer, car maker. Different terms can refer to the same concept.
5. "Unique id": may be modeled as an attribute (name), a property of the manufacturer class.
6. Make additional assumptions: Every manufacturer object *must* have an unique id.
7. 10001: attribute (value) of the id of a manufacturer object.
8. Name: a property of a manufacturer.
9. Additional assumption: Every manufacturer object must have a name.
10. Car: a class, as there may be many *brands* of cars.
11. Prepare questions: Do we need to introduce the concept *model* (e.g., Coupe, Sedan, Si Coupe)?
12. Civic and Accord: object instances of Car.
13. Additional assumption: Every car must have a name as its attribute.
14. Make, or manufacture: a relationship between a manufacturer (object) and a car (object).

**Class Diagram:**



**Object Diagram:**



**2.3 Classes**

1. Drawn as a rectangular box.
2. The class names, attributes, and operations may be specified, with selected details in the name, attribute, and operation *compartments* respectively.
3. Attribute and operation compartments are optional.
4. For DB modeling, the attribute compartment will eventually need to be clearly modeled. The operation compartments may not be needed.
5. The levels of details depend on the phases of modeling. It is a common mistake to specify too much detail in the early modeling phases.
6. As modeling proceeds, more details are added, updated, and refined.

Note that software application modeling and database modeling has different focus.

1. Software modeling: focus on operations (methods, especially public methods).
2. Database modeling: focus on attributes (data).

***Example:*** The following sequence of diagrams of how the modeling of the used car dealership application may proceed.

Initial version: v0.0.1.0:

* Only some major classes, associations, and attributes.

A diagram of a customer

Description automatically generated

Version v0.0.1.1:

* Add a payment class and some attributes.

A diagram of a company structure

Description automatically generated

Version v0.0.1.2:

1. Decided to split the concept 'car' into two concepts 'car model' and 'car'. Adjust associations.
2. Add some type information.

A diagram of a car model

Description automatically generated

Version v0.0.1.3:

1. Add an association between Payment and Car.
2. Add multiplicity of the association "of the model of":
   1. A car must be made of one car model.
   2. There may be many cars made of the same model.
3. Add multiplicity of 1 to the attributes Amount and PayTime of the class Payment. They are mandatory.

A diagram of a model

Description automatically generated

* For example, one may focus on the main classes and their associations in the first model, without worrying about the attribute or operation compartments.
* Most UML editors allow controlling visibility of different elements. For example, in Astah:



* A stereotype (specifying the kind of entities) and a property list with tagged values can be added to any compartment.
* Their flexibility allows for customization and extensibility to fit specific applications.
* Additional *properties*on data members may be specified, such as:
  1. Visibility: + (public: +, protected: #, private: -, etc.)
  2. data types
  3. abstract (in italic) or concrete (as constraints)
  4. class members (underscored) or instance members
  5. default values

**Example:** for software modeling:

A screenshot of a computer program

Description automatically generated

***Example:*** for database modeling.



The class Patron may be identified in the first draft of the UML class diagram.

1. In a subsequent iteration, attributes may be added using settings of the UML tool showing visibility of the attribute members.
2. Data types may be included using predefined data types provided by the tool.
3. In a further iteration, stereotype may be added, such as to identify the primary key <<PK>> and simple candidate key <<unique>>.
4. More specific user-defined types (or implementation types) may be used.
5. Operation members may be added. They are in general less important than data members in data modeling.
6. Multiplicity should eventually be added, as shown in the diagram below.



* Note that multiplicity can be used to depict *nullability* and *multi-valued* attributes. In this example, PatronId is not nullable ([1]), Phone is nullable ([0..1]) and Hobbies can have multiple values ([0..\*]).

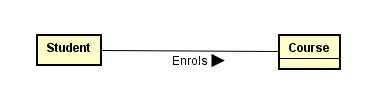
Check out the introductions to class diagrams from [agile modeling](http://www.agilemodeling.com/artifacts/classDiagram.htm) and [wikipedia](http://en.wikipedia.org/wiki/Class_diagram).

* Some *possible* relational database *extensions* on attributes may include:
  1. Multi-valued: \* or by using multiplicity.
  2. Multiplicity can also be used to indicate whether an attribute is nullable.
  3. Derived: <<derived>> using stereotype, \, or using other specific notation
  4. Primary key: <<PK>> as stereotype.
  5. Candidate keys: <<CK>> as stereotype.
  6. Unique field: <<unique>> as stereotype.
  7. Nullability: <<nullable>> or by using multiplicity.
  8. User-defined or system defined SQL data types.
  9. Indexing: <<index>> as stereotype.
* Check with your organizations for UML guidelines on a specific project.
* An example of a database profile for UML: <http://www.agiledata.org/essays/umlDataModelingProfile.html>
  1. may be adapted for uses in later phase of modeling.

**More Properties of Classes**

* A class is a 'first-class citizen.'
  + It has attributes.
  + It can form associations with other classes.
  + Objects of a class can exist by themselves.
  + It has more structures for modeling data requirements.
* As a comparison, an attribute is not a first-class citizen.
  + It does not have sub-attributes.
  + It cannot form associations with other elements.
  + The existence of an attribute depends on the object..
* Objects can be instantiated from classes.

***Example:***



We may have four objects of the student class: S1, S2, S3 and S4. Each student object represents an individual student in a database application.

We may have three objects of the course class: C1, C2, and C3. Each course object represents an individual course in the database application.

**2.4 Associations**

* Binary associations are represented by solid lines.
* Important options include:
  1. Association names with directional arrows (for reading).
  2. Association roles: the role of an object participating in an association.
  3. Multiplicities: the allowed number of associated objects.
  4. Association attributes can usually stored by promoting an association to an *association classes*.
  5. Qualifiers: association attributes to partition the targeted classes.
  6. Navigational requirement: specified by arrows. Usually not used in data modeling.
  7. Dependency constraints: by dotted lines.
* Some modeling questions and decisions:
  1. Should we model something as a class or as an association?
  2. Should we model something as a class or as an attribute?
  3. What kind of association should I use? Binary association, association class, n-ary association?

***Example:*** Note that no attribute is shown in this initial phase.

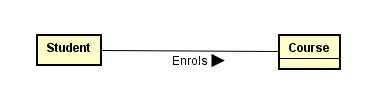
A diagram of a company

Description automatically generated

Note:

1. Job is an association class.
2. The arrow in the association "works-for" shows the direction of the association.
3. The association "manages" is between two job objects.
4. The {or} designation specifies the partition of the account class into two classes: person (account) and corporate (account).

***Example:*** For:



The association Enroll describes the association 'type'. An association is actually between two *objects* (a student object and a course object). Examples:

S1 -- C1: meaning student S1 is enrolled in course C1.  
S1 -- C3: (The associations S1--C2 and S1-C4 do not exist. This means the student S1 has not enrolled in the C2 or C4.)  
S2 -- C1  
S2 -- C2  
S2 -- C4  
S3 -- C3  
S4 -- C1  
S4 -- C4

**2.5 Multiplicity**

* Multiplicity can be specified by a number, the symbol \* (many), a range, or a set. Some example:
  + 0..1: zero or 1
  + 1..1: only 1
  + 1: may be 0..1 or 1..1; usually interpreted as 1..1
  + 0..\*: zero or many
  + 1..\*: 1 or many
  + \*: many; may be 0..\* or 1..\*
  + 1..4: 1 to 4
  + {1, 2, 6}: 1, 2 or 6
  + {1, 3:5, 7:9}: 1, 3, 4, 5, 7, 8, 9
* Multiplicity is a very common source of errors. Please refer to the explanation of the following diagram until you are very clear about it.

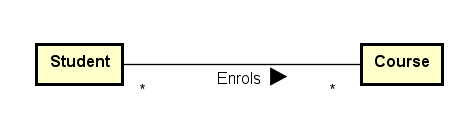


* Meaning:
  + *Every* X object must be associated with *n* Y objects.
  + Every Y object must be associated with m X objects.

***Example***

What do you think about these class diagrams?

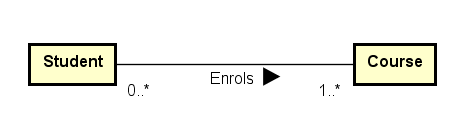
(a)



Assumptions made:

1. A student may take many courses.
2. Not sure whether a student is allowed to take zero course since \* (instead of 0..\* or 1..\* is used).
3. A course may have many students enrolled.
4. Not sure whether a course has no student enrolled since \* (instead of 0..\* or 1..\* is used).

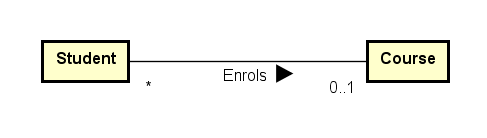
(b)



Assumptions made:

1. A student must be enrolled in one or more courses (may not be a reasonable assumption).
2. A course may have 0 or more students enrolled.

(c)



Assumptions made:

1. A student can only be enrolled in 0 or 1 course only (sound not reasonable).
2. A course may have many students enrolled.
3. Not sure whether a course has no student enrolled since \* (instead of 0..\* or 1..\* is used).

**Aggregation indicator**

1. aggregation (hollow diamond) and composition indicator (solid diamond):
2. Aggregation models the ‘a-part-of’ relationship (whole-part). E.g., car-wheel.
3. Composition is a *strong* form of aggregation: the part's lifecycle is dependent on the whole's lifecycle; e.g. university-department, building-room.
4. They can also be represented by using multiplicity.

***Example:*** Aggregation and Composition

A diagram of a graph

Description automatically generated

What do you think about this composition and aggregation examples in: [http://en.wikipedia.org/wiki/File:Congregationalism](http://en.wikipedia.org/wiki/File:AggregationAndComposition.svg)?

**Ternary Associations**

* N-ary associations are represented using a diamond connecting to participating classes.
  + Not so common.
  + May be modeled as a class instead.
* A ternary association involves three participating objects.

An example from a tutorial:

A diagram of a football team

Description automatically generated

Notes:

* In modeling, a ternary association can reasonably be replaced by a number of binary associations.
* Don't use n-ary associations where n>=3 unless you are sure.

**Generalization and Specialization**

* Generalization is represented by a hollow triangle at the superclass.
  + Generalization models the 'a-kind-of’ association.
  + It is mainly used to
    - manage classes with common data members and methods by put these common members into their superclass.
    - provide inheritance.
    - avoid multiple copies of member definition.
* Some options of generalization includes:
  + discriminator (the name of the partition),
  + powertype (a class in which an instance of it is a subclass of the superclass),
  + constraints (overlapping, disjoint, complete, incomplete and user defined constraints).

***Example:***

A diagram of a vehicle

Description automatically generated

A diagram of a tree

Description automatically generated

* There are many possible options and extensions.

**Constructing class diagrams: some tips**

1. There are many methodologies and best practice tips to construct effective class diagrams.
2. There are many possible modeling options: e.g., classes versus attributes, classes versus associations, multiplicity, etc.
3. Need to fully understand the assumptions and implications when making modeling decisions.
4. Do not model implementation details in earlier modeling phases.

**3. Example: toyu**

A reasonable conceptual model of the toyu database in UML:

A diagram of a computer program

Description automatically generated with medium confidence