**DASC 5333**

**2/4/2025**

Self Annotation

**The Relational Model**

by K. Yue

**1. Introduction**

**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
  2. Excel: table, and to be more exact, a two-dimensional array
  3. Object-oriented model: directed graphs
  4. XML: trees with many different types of nodes, plus sets of attributes.
  5. Cassandra: columnar or wild column model
  6. MongoDB: document model
  7. 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).
* E.g. toyu contains 8 tables.
  + A table contains many *rows* (tuples) and *columns* (attributes/fields).

A screenshot of a computer

Description automatically generated

* + Each row contains many *column values*.
  + Every row of a table has the same set of columns. (structured data)

A screenshot of a computer

Description automatically generated

* + Values of the same column have the same data *type*.
  + Keys are sets of columns.
  + A *candidate key* of a table is a *minimal unique identifier* of a row in the table.

E.g. For departments, two simple candidate keys:

1. deptCode (Primay key)
2. deptName (Alternate key)

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)

);

A screenshot of a computer

Description automatically generated

Student: CK: (1) stuId

Enroll table:

Primary key (PK)/CK: (1) {stuId, classId}: {} a set of two attributes/columns; minimal + unique; composite key (more than one attribute)

CREATE TABLE IF NOT EXISTS Enroll(

stuId INT NOT NULL,

classId INT NOT NULL,

grade VARCHAR(2) NULL,

n\_alerts INT NULL,

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

);

A screenshot of a computer

Description automatically generated

* + 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 references a primary key of another relation (known as the parent or referenced table).

Join tables: equality of common attributes: (FK = PK of the reference tables)

**Foreign keys:**  
  
1. Student(advisor) references Faculty(facId)  
2. Student(major) references Department(deptCode)  
3. Student(minor) references Department(deptCode)

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) UNSIGNED 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)

);

Q: StuId is the primary key of the database toyu?

No. Wrong question.

Right Q: StuId is the primary key of the table enroll? No, not unique

PK of enroll: {StuId, classId}

Q: {deptCode, SchoolCode} is a CK for department? False, not minimal. SchoolCode is not needed in the identifier.

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)  
11. Course(Rubric) references Department(deptCode)

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 not 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. It depends on problem requirements.
  3. A *relation schema*, R, is a *set* of attributes A1, A2,…,An with their domains D1, D2,…, Dn.
  4. A *tuple* (*row*) is a *set* of *mapping* of a *set* of attributes to a *set* of values: Ai -> di where di ∈ Di, for i = 1 to n (∈: belongs to)
  5. A *relation* (*instance*) is a set of tuples.
  6. 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 may be too restrictive for specific problems.

**1.3 Toyu Example**

[Toyu](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2025/notes/toyu/toyu.html): A drastically simplified university  
  
SQL statement: SELECT \* FROM class;

Select every column from the table class.

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 in be in the right order.

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.
    1. Because a relation is a set.
    2. Consequence: the relational model does not support 'object identity' directly.
  + The relational model is known to be '*value-oriented*':
    1. A row is a set of attribute values.
    2. Two rows with the same values in all attributes are the same row.
    3. 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. 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 (rows) of R have the same value of K.
  2. *Minimality*: No *proper* subset of K has the uniqueness property.
* Enroll table: CK: (1) {stuId, classId}: composite key
* A candidate key is a property of the semantic (meaning) of a relation.
* In other words, being 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 satisfies the uniqueness property.
  3. R or one of its proper subsets will satisfy the minimal property as extraneous attributes are removed.
* A set of attributes may be a candidate key of 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.
* 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.
* Some criteria for selecting a primary key:
  1. Stable and relative immutable.
  2. Simple: contains one attribute
  3. Concise: short storage space and faster comparison.
* An*alternate/secondary* key is a candidate key that is not the primary key.

A screenshot of a computer

Description automatically generated

Department: 2 CK

* deptCode: primary key
* deptName: secondary key/alternative 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. 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. No two departments can have the same name.
* deptCode is selected as the primary key of the relation department since it is concise and more stable.
* 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.
  3. A class can have many students.
* Note again that a candidate key is a property of a relation.

**2.1 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.
* 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.
* **Foreign keys:**  
    
  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)  
  11. Course(Rubric) references Department(deptCode)A foreign key may or may not have null value. It depends on the problem requirements.
* For uses in normalization theory:
  + An attribute that appears in one or more candidate keys is a *prime attribute* (or key attribute).
  + 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.)

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***: In the department table in toyu:

Department(deptCode, deptName, schoolCode, numStaff)

Candidate keys:

1. deptCode (i.e. {deptCode})
2. deptName

Primary key: deptCode

Alternate key:

1. deptName

Prime attributes:

1. deptCode
2. deptName

Non-prime attributes:

1. schoolCode
2. numStaff

Superkeys:

1. deptCode
2. deptName
3. deptCode, deptName (i.e. {deptCode, deptName})
4. deptCode, schoolCode
5. deptCode, numStaff
6. deptCode, deptName, SchoolCode
7. deptCode, deptName, numStaff
8. deptCode, SchoolCode, numStaff
9. deptCode, deptName, SchoolCode, numStaff
10. deptName, SchoolCode
11. deptName, numStaff
12. deptName, SchoolCode, numStaff

***Example:***

Study the list of all foreign keys in [toyu](https://dcm.uhcl.edu/yue/courses/joinDB/Spring2025/notes/toyu/toyu.html):

* 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)

***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?

**Null Values in SQL DB**

by K. Yue

**1. Null values**

* Generally used for representing missing information.
* SQL DBMS 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;  
  
-- 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)

**2. 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;  
    
SELECT \*  
FROM student  
WHERE 0;  
  
SELECT \*  
FROM student  
WHERE 1;  
  
SELECT \*  
FROM student  
WHERE 2697;  
  
SELECT \*  
FROM student  
WHERE '0';

-- ‘0’: string -> 0” tinyint (Boolean): implicit type conversion: convenient; inappropriate  
  
SELECT \*  
FROM student  
WHERE '145';  
  
-- 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)

**3. Interpretation of null values**

* Three possible interpretations of NULL:
  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 |  |

**Database Modeling**

by K. Yue

**1.Modeling and Modeling Transformation**

* Why modeling?
  1. Many DB developers may just directly construct the relational schema without formal data modeling. What are the problems with that?
     1. It may work for very simple problems.
     2. For more complicated problems, it is necessary to capture the detailed requirements before designing a solution.
  2. Initial ideas are usually based on needs. Their definitions can be very coarse and ambiguous. Real-world examples from past UHCL capstone projects:
     1. "I want a system to store and retrieve sale receipts for small business to claim sale tax waiver."
     2. "A mobile phone application to share and manage disaster recovery information."
  3. How do we construct a precise and executable computer solution to a vague real-world problem?

Software development is logical modeling (step [1] in the following diagram): constructing an *executable logical* model (such as using SQL, Java, etc., as the logical language.)

A diagram of a problem solving process

Description automatically generated

Logical modeling spans many phases in the software lifecycle:

1. Analysis
2. Design: architectural and component
3. Implementation (coding)

Furthermore, logical languages are closer to the machine world, and optimized accordingly. They are not designed to describe the real-world or capture real world requirement.

Conceptual modeling is introduced as shown in the diagram below:

A diagram of a problem solving process

Description automatically generated

Advantages of conceptual modeling:

1. Conceptual modeling languages, such as Unified Modeling Language (UML) or Entity-Relationship (ER) model, are designed to describe real-world problems and can better capture and refine problem requirements.
2. The process is a n-step process, breaking down the complexity.
3. Iterative refinement of the conceptual model provides eventually the necessary fidelity for different software development phases.

In Model-Driven Software Engineering (MDSE), software development is the development of a sequence of models, transforming higher level models to lower-level models in the process.

A diagram of a model

Description automatically generated

**2. Motivation for learning modeling**

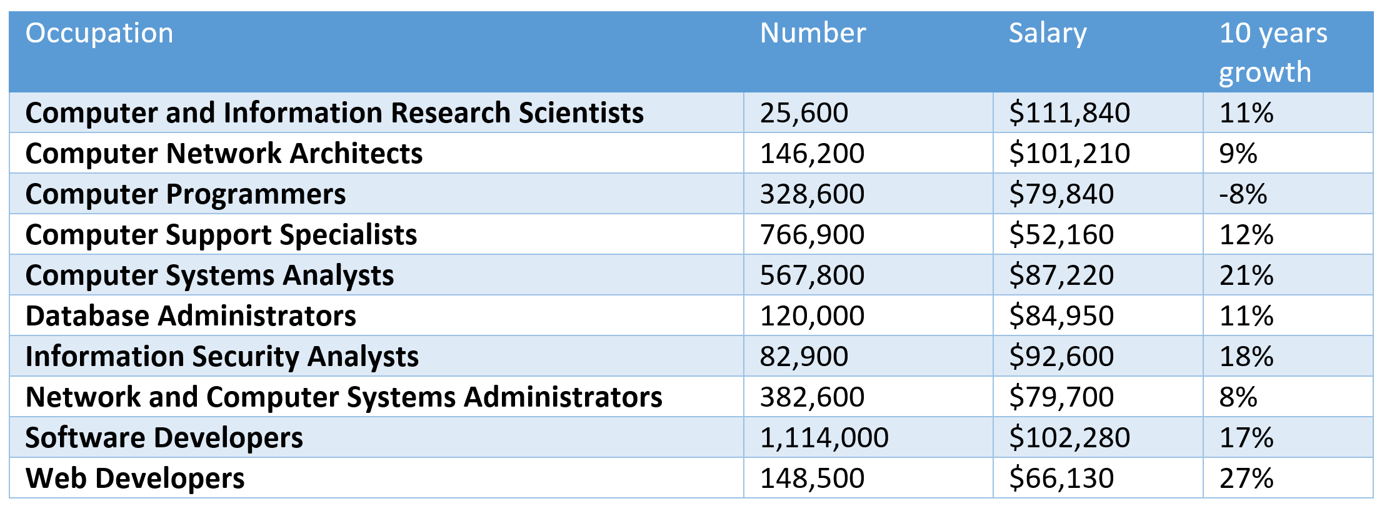
**Bureau of Labor Statistics, Computer and Information Technology Job outlook 2021 to 2031:**

A table of statistics with numbers and text

Description automatically generated

**Older data for comparison:**

Bureau of Labor Statistics, Computer and Information Technology Job outlook 2016 to 2026:



Average annual wage over all jobs in 2017: $37,600. 2016 to 2026 expected overall growth rate is 7%.

* A main difference between "software developers" and "computer programmers" are modeling and model transformation.

**3. DB Modeling**

**What is involved in DB modeling?**

1. Understand, capture, and refine the application requirements until they are clear and precise enough for design and implementation.
2. Communication is crucial. "Do*not* build *your* DB application. Build the DB application for the users."
3. Complexity management.
4. Three main topics in modeling tools and techniques:
   1. Modeling language: e.g., UML, ER.
   2. Theory and 'Good' practices: e.g., architectural and design patterns.
   3. Modeling process: what process to follow in modeling, e.g., Agile.

**3.1 Understanding the problem domain**

How do people solve problems?

1. Obtain an initial understanding of the problem.
2. Form a model to capture the understanding.
3. Refine the model to include necessary details.
4. Based on the model, devise a solution.
5. Implement the solution.

A serious problem of many novices is that they do not spend enough time and effort in steps (1) to (3) (i.e., modeling) and jump to (4) and (5) quickly.

* Your supervisor/user/customer/domain expert may not have a clear model themselves!
* In most cases, for novice software developers: what you think the applications should be is NOT as important! Instead, what the domain experts and the users think is crucial.

**What can you do to understand the problem and model it?** Some good practices:

1. Start by *asking a lot of questions*.
2. Collect and analyze as much documentation as possible.
3. Study and use the existing system: find out what is currently being used. Use them alongside the users. Participate in the work process.
4. Model, write, and document!
5. Use your own words and avoid the 'copy and paste' syndrome.
   1. A simple example below.
   2. The copy and paste syndrome in modeling and software development is much more difficult to unearth and costly to fix. A realistic example of an application letter to a Data Science Assistant Professor postion at UHCL.

A close-up of a letter

Description automatically generated

**Some possible generic questions to ask for modeling.**

1. What are the purposes of the application?
2. What problems will the application solve? What value will it provide?
3. What are the boundaries and context of the applications?
4. Who are the users?
5. Who are the domain experts?
6. What documentation is available?
7. How does the existing system work?
8. What are the workflow processes of the current system?

Prepare your own list!

**3.2 Modeling Languages**

**Two important options for database modeling languages:**

1. Entity Relationship (ER) Modeling and Extended ER (EER) Modeling: more relational database specific.
2. UML modeling: more general purpose.

**Some advantages of ER diagrams:**

1. Ease of understanding
2. Simpler
3. More specific to relational database modeling
4. Good collection of theory and best practices
5. Good vendor and tool support

**Some disadvantages of ER diagrams:**

1. Lower expressiveness
2. Not concise
3. Lack of standards
4. many versions that can be confusing
5. Narrow focus on relational databases

**What do we look for in a modeling language?**

1. Completeness: can it capture all needed information?
2. Expressiveness: can it capture information in an efficient and easy way for discussion and communication?
3. Domain support: how well does it support capturing the information in the application domain?
4. Precision: Is it detailed enough?
5. Accuracy: Is it ambiguous?
6. Conciseness: What is the 'information density'?
7. Standardization: Are there many competing versions?
8. Tool support: Are there strong tool support?
9. Community support: Are there a strong community?
10. Extensibility: Can it be extended to satisfy a specific domain?

We will use on UML in this course.

Read: A Badia and D. Lemire. [A call to arms: revisiting database design](http://dl.acm.org/citation.cfm?id=2070750). SIGMOD Record 40, 3 (November 2011), 61-69.

DOWNLOAD AND INSTALL ASTAH student version if you have a computer.

**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".

**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".
* A screenshot of a computer

  Description automatically generated

Example:

Problem description: A faculty member (e.g. Yue) advises a student (e.g. Smith) ….. A faculty member has a faculty id and name (breakdown to fname and lname)

Concepts (have data storage requirements):

* Faculty: modeled by a class (form objects, e.g. Yue, Hasan, Wu, …)
* Student: modeled by a class.
* Advises: modeled as an association: Yue advises Smith
* Faculty id is a property of a faculty (e.g. Bun Yue’s id is 1234567): attribute of facully objects

A screenshot of a computer

Description automatically generated

Example of creating the class ‘class’:

A screenshot of a computer

Description automatically generated

2025\_2\_4\_ex1.asta:

Class:

1. Name
2. Data member/attribute compartment: properties
3. Methods/Operation compartment: less important in data

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

* 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. Study 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 any 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 a unique id and name. A car maker may make many cars. For example, Honda, which may have a manufacturer id of 10001, makes Civic and Accord.

Concepts:

1. Car: class
2. Manufacturer: class
3. Unique id: 10001 (X) is the id of Honda (Y): attribute of manufacturer
4. Name: attribute of manufacturer
5. Car maker: synonym of manufacturer; no need to model.
6. Make
7. Honda
8. Manufacturer id
9. 10001
10. Civic
11. Accord

Choices of model decisions:

1. Class X (object: instance of a class); have properties, associations. Objects exists by themselves X by itself. E.g. X: faculty; Yue faculty object
2. Attributes (properties) X of a class, etc: value only; simple. Fname of a faculty. X of Y. Bun is the first name of the faculty ‘Bun Yue’
3. Associations X (relationship between two objects). A – X – B. For example Yue (A) advises (X) Smith (B)
4. No need to model

**Analysis and Design**

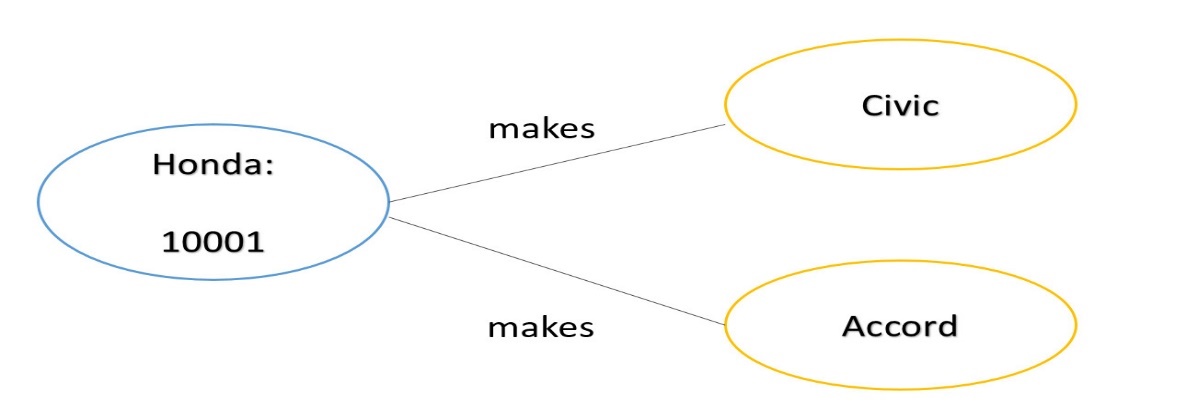
Some observations:

1. Manufacturer: a class (template) that can be used to initiate many manufacturer objects (instances).
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 (i.e., a manufacturer object such as Honda).
4. Define synonym: manufacturer, car manufacturer and car maker may be the same. 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: E.g., 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. Another additional assumption: Every manufacturer object must have a name.
10. Car: a class, as there may be many *brands* of cars.
11. Prepare questions: E.g., 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). A manufactor (object) makes (verb: possible asspication) a car (object).

**Class Diagram:**



**Object Diagram:**



**2.3 Classes**

**2.3.1 Basics**

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,
   1. The attribute compartment will eventually need to be clearly modeled.
   2. The operation compartment 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 have different foci.

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 classes Patron, Member and Department with some attributes may be modeled in the first draft of the UML class diagram. Boxes in the diagram above:

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 for Patron below.



* Note that multiplicity can be used to depict *nullable* 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 notations
  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.

**2.3.2 More Properties of Classes**

* A class is a 'first-class citizen.'
  + It has attributes.
  + It can form associations with other classes.
  + It can have operations.
  + 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 in the following diagram until you are very clear about it.

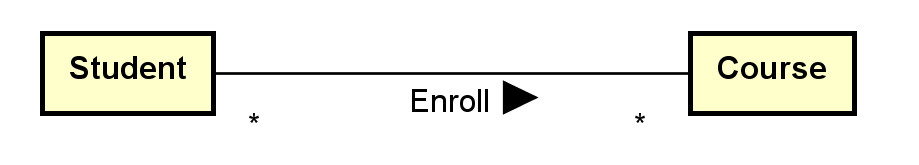


* Meaning:
  + *Every* X object must be associated with *n* Y objects.
  + Every Y object must be associated with m X objects.

***Example***

What do you think about these class diagrams?

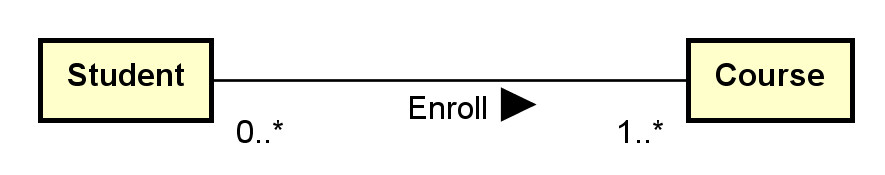
(a)



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).

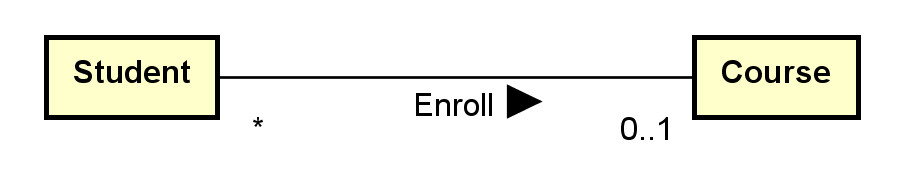
(b)



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 promoting it to a class and add three 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 putting these common members into their superclass.
    - provide inheritance.
    - avoid multiple copies of member definition.
* Some options of generalization include:
  + 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

* 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. Study 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 any 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 a unique id and name. A car maker may make many cars. For example, Honda, which may have a manufacturer id of 10001, makes Civic and Accord.

**Analysis and Design**

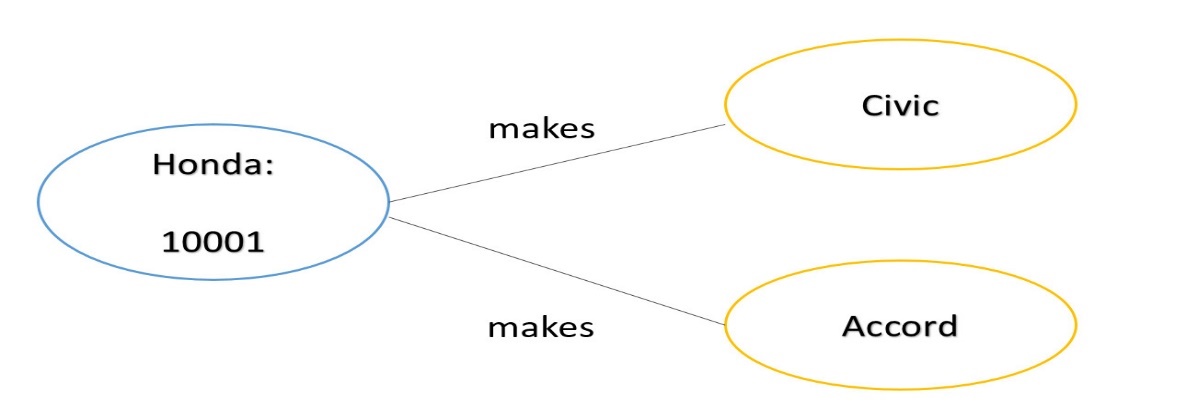
Some observations:

1. Manufacturer: a class (template) that can be used to initiate many manufacturer objects (instances).
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 (i.e., a manufacturer object such as Honda).
4. Define synonym: manufacturer, car manufacturer and car maker may be the same. 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: E.g., 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. Another additional assumption: Every manufacturer object must have a name.
10. Car: a class, as there may be many *brands* of cars.
11. Prepare questions: E.g., 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). A manufactor (object) makes (verb: possible asspication) a car (object).

**Class Diagram:**



**Object Diagram:**



**2.3 Classes**

**2.3.1 Basics**

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,
   1. The attribute compartment will eventually need to be clearly modeled.
   2. The operation compartment 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 have different foci.

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 classes Patron, Member and Department with some attributes may be modeled in the first draft of the UML class diagram. Boxes in the diagram above:

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 for Patron below.



* Note that multiplicity can be used to depict *nullable* 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 notations
  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.

**2.3.2 More Properties of Classes**

* A class is a 'first-class citizen.'
  + It has attributes.
  + It can form associations with other classes.
  + It can have operations.
  + 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 in the following diagram until you are very clear about it.

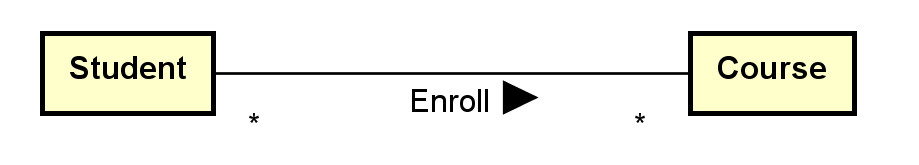


* Meaning:
  + *Every* X object must be associated with *n* Y objects.
  + Every Y object must be associated with m X objects.

***Example***

What do you think about these class diagrams?

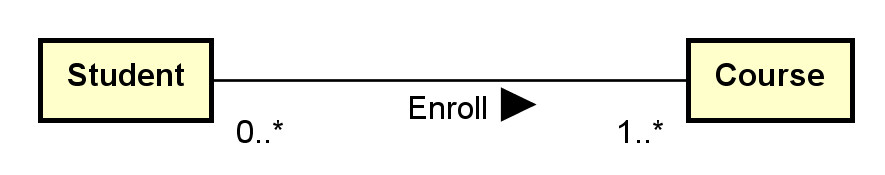
(a)



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).

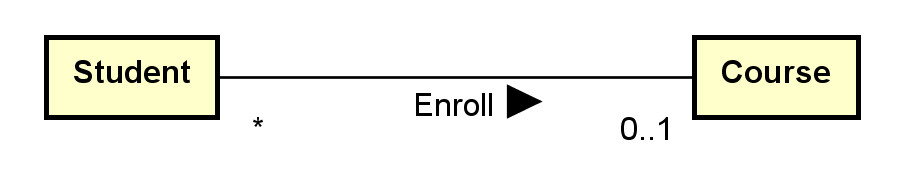
(b)



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 promoting it to a class and add three 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 putting these common members into their superclass.
    - provide inheritance.
    - avoid multiple copies of member definition.
* Some options of generalization include:
  + 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