**CSCI 4333.1**

9/4/2024

**Database Basics**

by K. Yue

**1. Introduction**

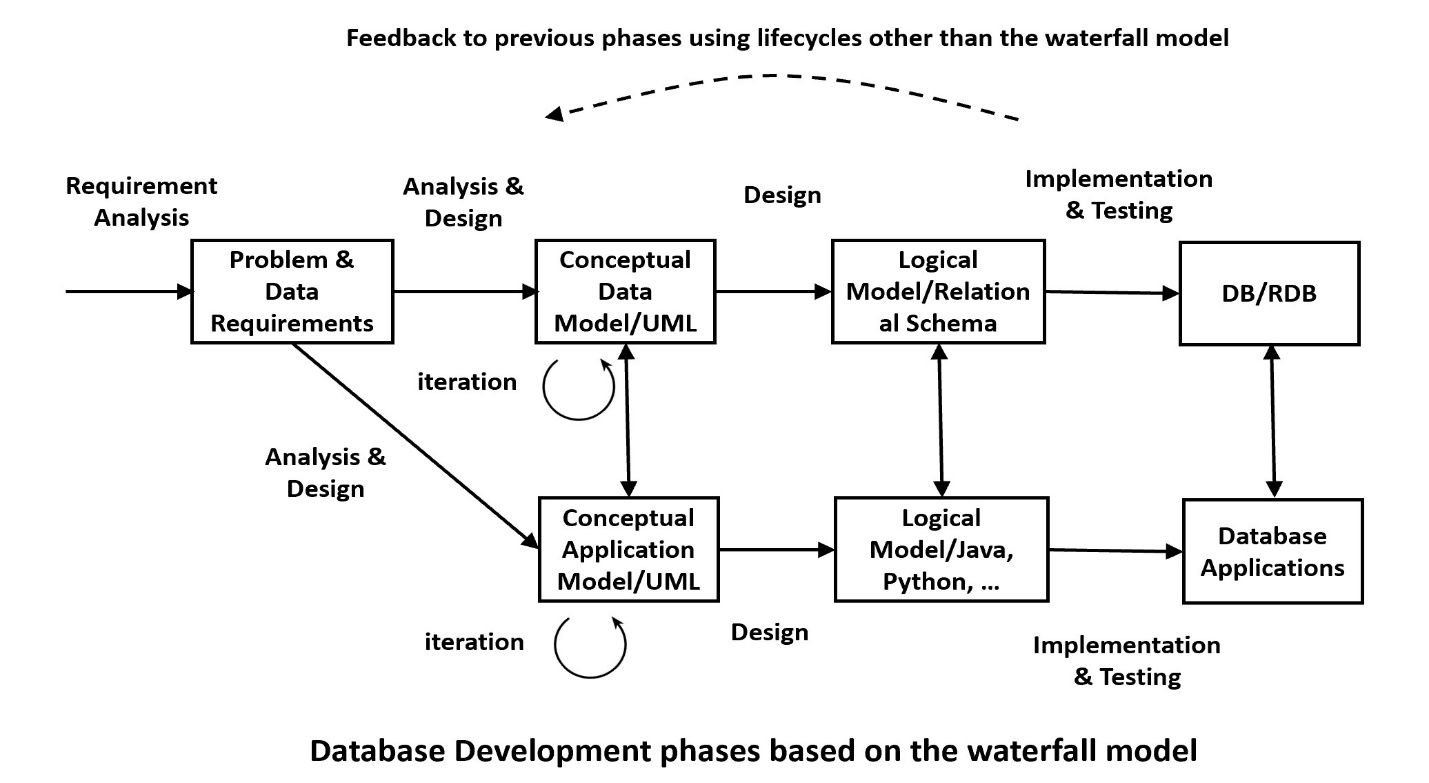
* It is important to be familiar with the basic terms and concepts of databases in this course.
* A database system is built by using a Database Management System (DBMS).
* One popular DB engine ranking: <http://db-engines.com/en/ranking>.
* We focus on Relational DBMS (RDBMS).
* Examples of Relational DBMS:
  1. Access: most popular 'departmental' DBMS
  2. Oracle: Most popular commercial DBMS
  3. MS SQL server: Likely second most popular commercial DBMS
  4. MySQL: most popular open source DBMS
  5. MariaDB: highly compatible to MySQL (more open source than MySQL)
  6. Postgres: popular open source DBMS known for innovation and functionality.
  7. SQLite: most popular portable DB engine.
* Relational DBMS basically use the relational model (with extensions).
* There are many other models. Examples:
  1. Object-Oriented Database (OODB): e.g., db4o, Gemstone, etc.
  2. Big Data:
     + Document DB: e.g., MongoDB, CouchDB
     + Key-Value DB: e.g., Redis, LevelDB
     + Wide Column DB: optimized over large dataset; store columns together, not rows. E.g. Cassandra and HBase.
     + Graphical DB: e.g., Neo4J

**2. Users**

* Users drive requirements. It is always important to find out the types of users.
* There are many types of users in a RDB. Three major kinds:
  1. End users: usually do not use SQL to access the database directly. Examples:
     + front-end users
     + managers and staff
     + domain experts
  2. DB application developers and data analysts
     + Develop DB solutions using SQL.
     + With various levels of access privileges.
  3. DB administrators
     + Manage the entire DB, such as:
       - conceptual and physical database design and implementation
       - security
       - user account management
       - backup and recovery
       - performance tuning
* The likely role of most of you: DB and application developers and data analysts.
* You are likely not the end users of the DB system you built.
* Thus, do *not* build the database for yourself.

**3. DB Development Phases**

* The classical waterfall software development life cycle can be useful as a basis to understand the various phases of database development.
  1. Requirement: conceptual modeling.
     1. Planning
     2. Analysis
  2. Design:
     1. Logical modeling and design
     2. Physical Design
  3. Implementation
  4. Testing
  5. Maintenance
* There are many other software lifecycle models.
* DB development is a kind of software development.



**3.1 Data Modeling**

General conceptual model. See, for example: <https://en.wikipedia.org/wiki/Conceptual_model_(computer_science)>.

1. Capture domain knowledge and requirements from the business and application perspectives.
2. Driven by requirements.
3. Construct a conceptual model iteratively.

See, for example: <https://en.wikipedia.org/wiki/Conceptual_schema>.

Diagram of a model architecture

Description automatically generated

**Identify and capture user requirements:**

1. Likely the most tedious and difficult parts for many traditional applications.
2. Collect documents of existing systems.
3. Study documents of existing systems.
4. Talk with domain experts and end users.
5. Model the problem using a modeling language, such as UML, ER, etc.
6. Document the captured requirements: e.g., modeling, requirement specifications, data dictionary, etc.
7. Iteratively refine and correct the model until enough details are captured.

**3.2 Design database solutions**

1. Select the appropriate data model of the database.
2. Select the appropriate DBMS.
3. Design the *logical* model.
4. Design the architecture of the DB system.
5. Design the physical database.
6. Design external views.
7. Design individual components.

**3.3. Implementation and testing**

1. Implement and test design.
2. Optimize performance.

**4. The Three-Layered DB Architecture**

* The three-layered database architecture is well known and you can get a lot of information about it from the Web. Examples:
  + A simple one: <https://www.tutorialspoint.com/Three-Level-Architecture-of-Database>
  + A more nuanced one: <http://jcsites.juniata.edu/faculty/rhodes/dbms/dbarch.htm>
  + Consult Figure 2.4 of Ricardo.
* Use the *layer* pattern to manage complexity. The *layer pattern* is an important concept in Computer Science and software architecture.
* Three levels:
  + External or view level: Describes a part of the database for a particular user group, Provide the right level of abstraction and security control.
  + Logical level: Describe logical structure of the entire database.
    1. Some practitioners call the 'logical level' the 'conceptual level'. This can cause confusion as other distinguish between 'conceptual model' and 'logical model'.
  + Internal/Physical level: Describe physical storage structure of the database.
* Provide data independence:
  + Logical data independence:
    1. between logical database and external views.
    2. Changes in the logical database may not affect the external views.
  + Physical data independence:
    1. between logical database and physical database.
    2. Changes in the physical database do not affect the correctness of the logical database.
* The logical level is the focus.

A diagram of a data base

Description automatically generated

Logical level: e.g. toyu, SQL

A screenshot of a computer

Description automatically generated

Physical DB:

A screenshot of a computer

Description automatically generated

**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/set 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*.
  + Keys are sets of columns. (set-theoretic model)
  + A *candidate key* of a table is a *minimal unique identifier* of a row in the table.
* In Enroll,
* The CK: {stuId, classId}
* 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
* );
* Uniqueness: {stuId, classId} uniquely identifies a row.
* Minimality:
  + stuId does not uniquely identify a row.
  + classId does not uniquely identify a row.
  + A *primary key* is a selected candidate key (for storing the table).
  + *Alternative/secondary keys* are candidate keys not selected as the primary key.
* School table:
* 2 CK: [1] schoolCode: PK, [2] SchoolName: secondary CK.
* CREATE TABLE IF NOT EXISTS School (
* schoolCode CHAR(3) NOT NULL,
* schoolName VARCHAR(30) NOT NULL,
* CONSTRAINT School\_schoolCode\_pk PRIMARY KEY (schoolCode),
* -- alternate keys: [1] schoolName
* CONSTRAINT School\_name\_ck UNIQUE (schoolName)
* );
  + A *foreign key* of a relation references a primary key of another relation (known as the parent or referenced 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 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.
* E.g. R: student; A1 : stuId; D1 : INT
* 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 ∈ Di, for i = 1 to n (∈: belongs to)
* 
* Row: stuId -> 100000; fname -> Tony, …
  1. A *relation* (*instance*) is a set of tuples.
* A screenshot of a computer

  Description automatically generated
  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 may be too restrictive for specific problems.

**1.3 Toyu Example**

[Toyu](https://dcm.uhcl.edu/yue/courses/joinDB/Fall2024/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 of R have the same value of K.
  2. *Minimality*: No *proper* subset of K has the uniqueness property.
* 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.
* 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.
* 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/Fall2024/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.
* Null is a built-in value.
* 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’ -> 0: implicit type conversion (string -> tinyint)

Strongly typed language -> less implicit type conversion.

Rules: be as specific and as explicit as possible.  
  
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 |  |