Preface:

This is a tutorial about Introduction to Database system(cpt103_2324fall) for students in xjtlu. I revise some contents in ppt that have false logic , make up some contents from gpt-3.5 , translate some contents and add some code.Please feel free to contact me in email: nakupenda.ics@gmail.com . Also, I need to admit that this is just a basic understanding for database. I recommend to continue study: Introduction to Database system (CMU15.445(645)_22fall) and I will update my note sooner or later.

Introduction to Database system: xjtlu_cpt103_2324fall

0 Introduction to Database Systems & Relation model & Relation key

1 What is data and database?

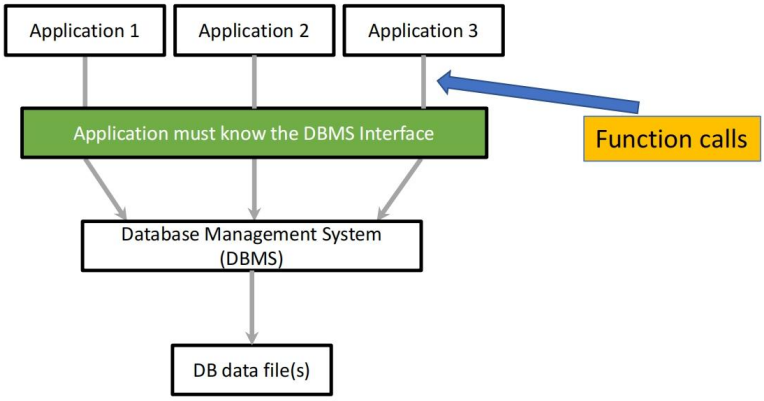
• Data is only meaningful under its designed scenario.
----Must have ways to create/modify data.
----Must have ways to access data.
• Database: Organised 有组织的 collection of data. Structured, arranged for ease 方便 and speed of search and retrieval 检索.
• Database Management System (DBMS): Software that is designed to enable users and programs to store, retrieve and update data from database.
----A software must have a set of standard functions to be called DBMS.

2 Previous Methods before DBMS appearing

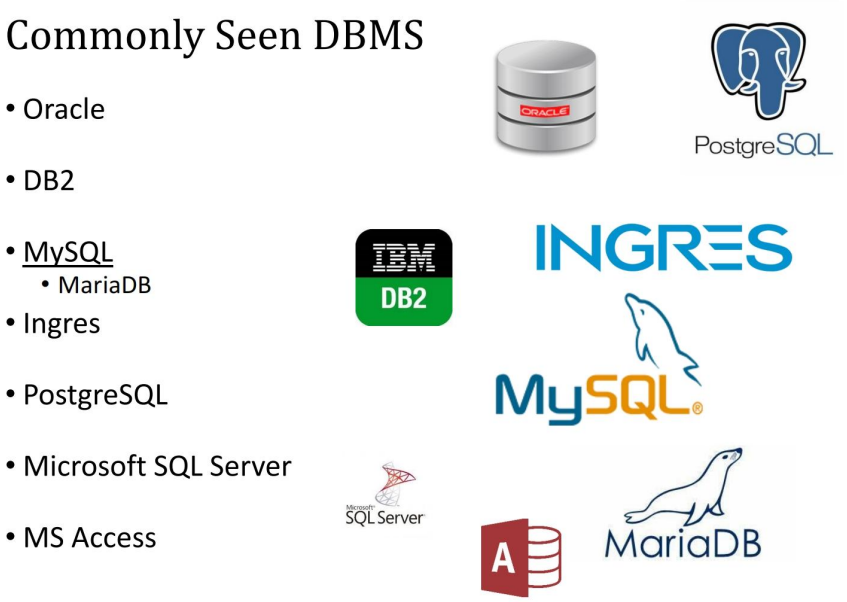
• Applications store data as files.
---- Each application uses its own format.
• Other applications need to understand that specific format.
---- Leads to duplicated code and wasted effort.
---- Compatibility 兼容性 issues.
•How about using a common data format?
---- Still need to write code for reading this file format.
---- Synchronisation 同步 issues: Accessed simultaneously?
---- Very hard to coordinate operations from different apps.
-----Compatibility issues.

3 DBMS approach

- Work as a delegate for this common collection of data.
- Applications use a common API for accessing database.
  - The implementation of API is provided by database software companies.
  - All database commands are standardised (SQL language).



4 Commonly DBMS



5 DBMS functions

- Allow users to store, retrieve and update data
- Ensure either that all the updates corresponding to a given action are made or that none of them is made (Atomicity)
- Ensure that DB is updated correctly when multiple users are updating it concurrently
- Recover the DB in the event it is damaged in any way
- Ensure that only authorised users can access the DB
- Be capable of integrating with other software

6 The Relational Model

• The relational model is one approach to managing data.
----Originally Introduced by E.F. Codd in his paper "A Relational Model of Data for Large Shared Databanks", 1970.
----An earlier model is called the navigational model
• The model uses a structure and language that is consistent with first-order predicate logic 一阶谓词逻辑
---- Provides a declarative method for specifying data and queries
• Relational database management systems (RDBMS) are based on the relational model.
---- Many relational operations are supported.
---- Relational algebra

关系模型使用结构和语言与一阶谓词逻辑一致。它基于关系代数和关系演算，提供了一种声明式的方式来定义数据结构和查询。

在关系模型中，数据被组织为关系（表）的集合，每个关系包含元组（行）和属性（列）。关系之间通过键和外键建立关联。关系模型的核心思想是通过关系操作（如选择、投影、连接等）来操作和查询数据，而不需要关注底层数据的存储细节。

7 Terminologies 专用术语

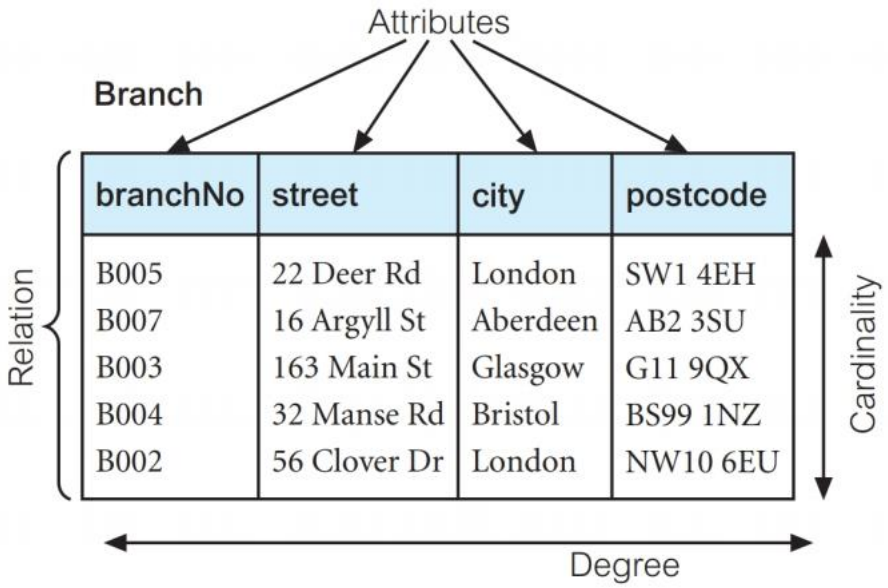
• A relation is a mathematical concept. The physical form of a relation is a table with columns and rows.

• An attribute 属性 is a named column 列 of a relation.

• A domain is the set of allowable values for attributes. Example:

----Age must be a positive integer.

----Postcodes have length limit



• Tuple 元组: a tuple is a row of a relation.在关系模型中，元组是关系中的一行数据。它表示了一个实体或对象的特定实例。每个元组包含一组属性值，对应于关系模式中定义的属性（列）

• Mathematically, the order of tuples does not matter.

• Degree: the number of attributes it contains.

• Cardinality 基数: the number of tuples in a relation

• Relation schema: The definition of a relation, which contains the name and domain of each attribute. ,"A named relation defined by a set of attribute and domain name pairs"

Table: branch

branchNO	Character: size 4, range B001-B999
street	Character: size 25
city	Character: size 15
postcode	Character: size 8

• *Relational database schema:*
---- A set of relation schema, each with a distinct name.
----Could be understood as a set of table definitions like the above example

• *Alternative terminology*

Formal Terms	**Alternative #1**	**Alternative #2**
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

• *Example:*

Staff

ID	**Name**	**Salary**	**Department**
M139	John Smith	18000	Marketing
M140	Mary Jones	22000	Marketing
A368	Jane Brown	22000	Accounts
P222	Mark Brown	24000	Personnel
A367	David Jones	20000	Accounts

Attributes : ID, Name, Salary & Department
Relation schema:
relation_name(ID: Char, Name: Char, Salary: Monetary 货币, Department: Char)
Tuples, e.g.row3:
{(ID, A368), (Name, Jane Brown), (Salary, 22,000), (Department, Accounts)}
Degree:4
Cardinality:5

8 Additional Properties of Relations

- *Relation's name is unique in the relational database schema.*
- *Each cell contains exactly one atomic value.每个单元格只包含一个原子值。*
- *Each attribute of a relation must have a distinct name.*
- *The values of an attribute are from the same domain.*
- *The order of attributes has no significance.*
- *The order of tuples has no significance.*
- *No duplicate tuples 不能出现重复完全一致的一行*

9 Relational Keys

- *Assume each person's id is unique.*
- *Assume that the whole relation is stored as a twodimensional array, and you want to look for the 'Maria' whose age is 22.*
- *What problem does the relation below have?*
- *How many rows do you need to check?*
- *What can be done to improve the efficiency of this search and prevent this from happening?*

ID	Name	Age
1	Andrew	34
1	Andrew	34
1	Andrew	34
2	Erick	32
2	Erick	32
3	Thomas	28
4	Paul	33
6	Rodrick	47
7	Maria	55
8	Maria	22

9.1 Primary Key 主键

- *It is beneficial to let a program to automatically check for and reject duplicate values in one or more columns for you when tuples are added.*
- *This can be done in database systems, by applying a constraint 限制 (consider it as a label) called Primary key on the columns of a table.*

- *Single-column primary key example (Staff table):*

ID	Name	Age
1	Jason	12

• *Multi-column primary key example (Company with several buildings):*

Building Number	Room	Room Size	Has Printer
11	301	96	True

主键是用于唯一标识关系（表）中每个元组（行）的属性或属性组合。主键的值在关系中必须是唯一的，每个元组都必须具有唯一的主键值。这意味着没有两个元组可以具有相同的主键值。

• *Columns constrained by a primary key uniquely identifies tuples in a table.*
---- *For each tuple, the id is always different.*
---- *This is a core functionality of primary key.*
• *Each table can only have one primary key.但是，一个主键可以由一个或多个列组成。这称为复合主键。复合主键是由多个列的值组合而成，用于唯一标识每个行。复合主键允许我们在多个列上定义主键约束，并确保这些列的组合是唯一的。*
• *NULL values are not allowed if a primary key is present.*

• *Primary Key enforces entity integrity 实体完整性.*
• *It helps to maintain the consistency and accuracy of data in a database by preventing duplicate records and ensuring that each record can be uniquely identified.*
• *Particularly important in applications that rely on a high degree of data integrity:*
---- *Financial systems*
---- *Healthcare applications*
---- *Other mission-critical systems.*

9.2 Super Key 超键

• *Super key: using more than enough columns to uniquely identify tuples in a table.*
---- *In upper table, only the constraint (ID) is a primary key.*
---- *(ID, Name), (ID, Age), (ID, Name, Age) are super keys.*
---- *(Name), (Name, Age) are bad choices of primary keys.*
• *Super key is taught so that you can avoid them when choosing the columns to be applied with primary keys*
主键是超键的子集

• *When we explained how databases check for duplicate values, it was in an linear way. In reality, the check will be faster. But even if it is faster, super keys are still bad for performance as more comparisons are needed when checking for duplicate values or looking for a certain value.*
• *You need a good understanding of data structure to understand the underlying mechanism.* ***BTree + Primary key***

9.3 Candidate Key 候选键

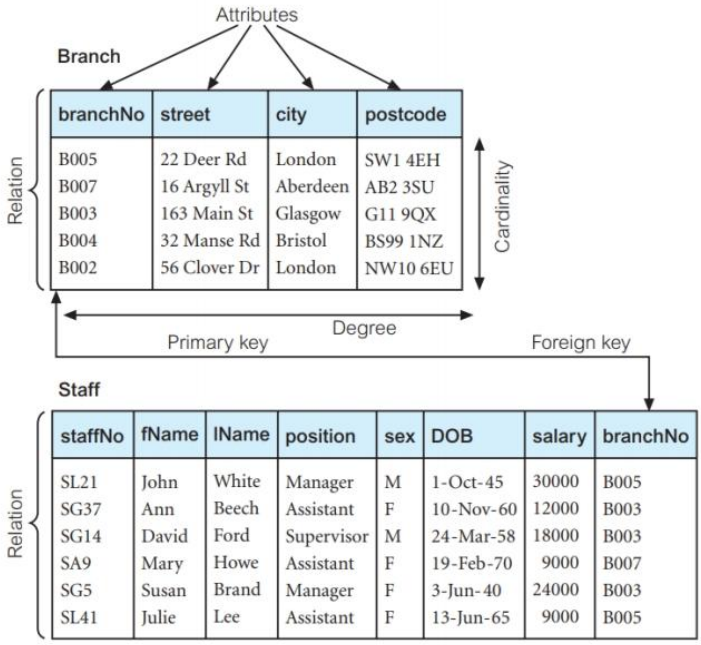
StaffID	Email	First name	Last name	Passport ID
1	S.Guan@xjtlu.edu.cn	Steven	Guan	P123456
2	J.Woodward@nott.ac.uk	John	Woodward	U543121
3	N.Tubb@bhan.ac.uk	Nathan	Tubb	U998877

• All these possible primary keys are called Candidate keys.
---- The primary key is just a candidate key chosen by the table designer.
----There's no deterministic way to determine which candidate key should be a primary key.
• Sometimes,you can't infer the candidate keys based on the data in your table
----More often than not, an instance of a relation will only hold a small subset of all the possible values
----E.g. Restaurants' booking number might reset to 1 after a large number.

候选键是可以唯一标识关系表中每个记录的一组属性（列）。候选键的特点是它们的属性组合能够保证在表中没有重复的记录。一个关系表可以有多个候选键，但通常选择其中一个作为主键（Primary Key）来标识唯一性。主键是从候选键中选择的一个，用来唯一标识表中的每个记录。

9.4 Foreign Key

• It is also very common that tuples in one relation references data from another relation.As a result, a database should provide such mechanism to ensure correct references. This is enforced by something called foreign key



• *Foreign key: One or more attributes within one relation that must match the candidate key of some (possibly the same) relation.。*

----外键并不要求数据必须是唯一的，但它要求外键值在父表的候选键中存在。在子表中的外键列上的值可以重复，但这些值必须在父表的候选键中存在。外键可以存在空值。

----外键列上的空值表示该行数据没有对应的父表记录。如果父键被定义为 NOT NULL，外键依然可以是 NULL。如果在外键列上使用了 NOT NULL 约束，则该列不允许为空值。在这种情况下，您必须插入一个有效的外键值，否则触发外键约束错误。请尽可能地避免在外键列上使用 NULL 值，可能会导致查询结果不准确或性能下降。因此，尽可能地定义外键列为 NOT NULL，并在插入数据时提供有效的外键值。

• *Data type should be the same.*

---- In real databases, sometimes this can be violated.

---- Different data type is not recommended.

• *Foreign Key enforces referential integrity*

---- It ensures that all data in a database remains consistent and up to date.

---- It helps to prevent incorrect records from being added, deleted, or modified.

1 SQL: Tables and Data

1 SQL Background

• ***SQL ==Structured Query Language**. which consists of two parts:*

----Data definition language (DDL).

----Data manipulation language (DML).

*----SQL statements will be written in **BOLD COURIER FONT***

----SQL keywords are not case-sensitive 不区分大小写

---- SQL statements should end with semi-colons 分号.

----Two dashes followed by a space will comment out 注释 that line

```
-- commented out
--not correctly commented out
```

• *Database Containment Hierarchy*

---- A computer may have one or more clusters 集群 (another name for database server)

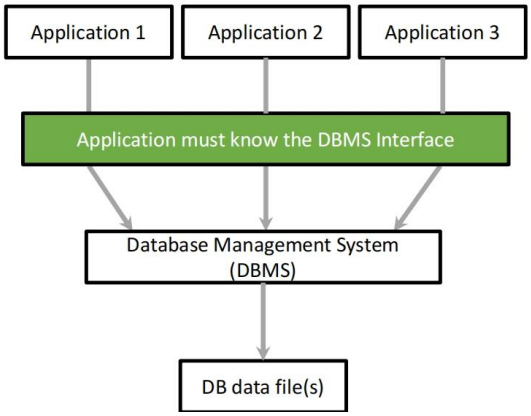
---- A computer can run multiple clusters.一个计算机可以运行多个数据库服务器，这意味着它可以同时承载多个数据库实例或服务。每个数据库服务器可能使用不同的数据库管理系统（例如 MySQL、Oracle、SQL Server 等），并且可以独立地管理和提供数据库服务。

---- Each cluster contains one or more catalogs (another name for database).

---- Each catalog consists of set of schemas,and a schema consists of tables, views, domains, assertions, collations, translations, and character sets. All have same owners.

SQL Database and Related Software

- DBMS is an application that usually runs on a server.
  - MySQL, MariaDB, PostgreSQL, Oracle…
- Different client applications can access this server simultaneously.
  - DBeaver, MySQL workbench, PhpMyAdmin
  - Your own programs.



• *How to start? password----cx20040120*

---- First, we need to create a schema

CREATE SCHEMA name; == CREATE DATABASE name;

---- If you want to create tables in this schema, you need to tell MySQL to "enter" into this schema, type: ***USE name;***

----Then all following statements like ***SELECT*** *or* ***CREATE TABLE****, in the same script, will be executed in this schema. Just like a file manager, after entering a folder, all operations will apply to this folder*

----Avoid creating or modifying tables in these databases that are created automatically by the system: information schema/ mysql/performance schema/ sys , as they are created for database administration purpose

---- Error messages from SQL server are not always informative.提供信息

---- Being able to check the official manual is a very important skill.The Official Manual: https://dev.mysql.com/doc/refman/8.0/en/

2 SQL: Creating Tables Code: DBL1 提供特别多的细节问题不在这里展示

• *To create tables in SQL, you need the* ***Create table*** *statement, the general syntax is shown below:*

```
CREATE TABLE [IF NOT EXISTS] name (
    col-name datatype [col-options],
    .....
    col-name datatype [col-options],
    [constraint-1],
    .....
    [constraint-n]
);
```

• *In SQL, table or column names can have spaces, but it is strongly not recommended.*
• *If you insist to do so, you must enclose the name with a pair of (`) symbols.*

```
CREATE TABLE mytable (
      columnA INT,
      columnB CHAR(11)
);

CREATE TABLE `my table` (
      `column A` INT,
      `column B` CHAR(11)
);
```

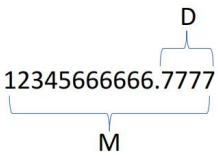
2.1 Integers

• ***SMALLINT***
----Use 2 Bytes of memory
----range: -32768 to +32767
• ***INT or INTEGER***
----Use 4 Bytes of memory
----range: -2147483648 to +2147483647
---- A typical choice for integer.
• ***BIGINT***
---- Use 8 Bytes of memory.

```
CREATE TABLE `staff` (
      `name` VARCHAR(12),
      `staff_id` INT
);
```

2.2 Fixed Point

• ***DECIMAL[(M[,D])] or NUMERIC[(M[,D])]***
---- Fixed point number. Things between [] are optional
----Decimal(size,d)




```
CREATE TABLE `staff` (
      `name` VARCHAR(12),
      `staff_id` INT(11),
      `salary` DECIMAL(5,2)    --  数据类型为5 位数字其中2 位是小数
);
```

2.3 Float

• **FLOAT(p)**

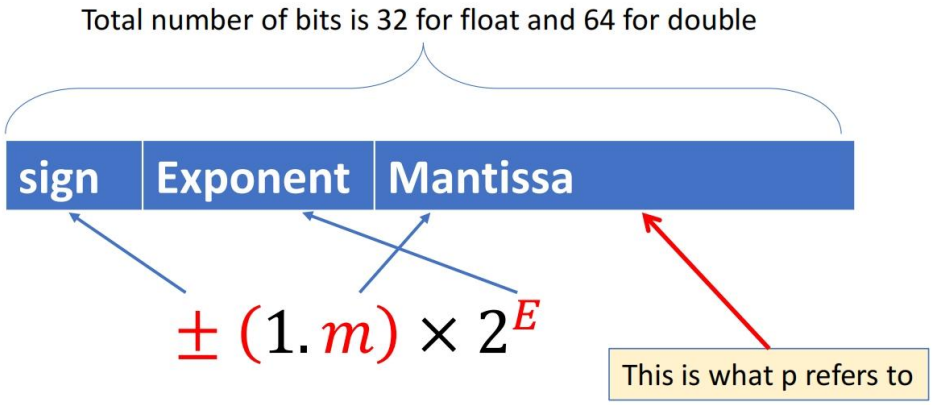
----p 参数用于指定浮点数的精度。它表示浮点数的位数，即尾数的大小。例如，FLOAT(6)表示尾数有 6 位。

----Floating point number in IEEE 754 standard

----prepresents the precision in bits. (Mantissa size)

---- MySQL will automatically choose single precision 单精度（32 位） or double precision 双精度（64 位） based on the value of p.

----请注意，浮点数是近似值，不适合用于精确计算，因为它们可能会导致舍入误差。如果需要进行精确计算，建议使用 DECIMAL 类型而不是 FLOAT 类型。



2.4 String

CHAR[(M)]

----A fixed-length (M is the length, 0 ~ 255) string

----Always right-padded with spaces to the specified length when stored.存储时总是用右填充指定长度的空格。

----E.g. if you store 'A' to CHAR(5), it will actually be 'A ' (4 个空格）inside the memory. But when you retrieve the value, the trailing spaces will be removed automatically (this behaviour can be turned on or off).

VARCHAR(M)

---- A variable-length string.

---- The range of M is 0 to 65,535.

Comparison For MySQL:

---- CHAR is faster, but occupies more memory

---- VARCHAR is slower, but occupies less memory

• String values in SQL are surrounded by single quotes:

---- 'I AM A STRING'

• Single quotes within a string are doubled or escaped using \ 字符串中的单引号将加倍或使用 \： *'I''M A STRING'* *'I\'M A STRING'* *'' this is an empty string*

• *In MySQL, double quotes also work (Not a standard)*
• *Example (after creating the table)*

```
CREATE TABLE `staff` (
    `name` VARCHAR(12),
    `id_card` CHAR(6)
);
insert into `staff` values ('Daryl', 'STF001');
```

2.5 Date and Time

• ***DATE***

----The supported range is '1000-01-01' to '9999-12-31'.
----MySQL displays DATE values in 'YYYY-MM-DD' format.

• ***DATETIME[(fsp)]***

---- The supported range is '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999'
---- Display format in MySQL: 'YYYY-MM-DD hh:mm:ss'
---- fsp 代表 fractional seconds precision 小数秒精度

```
CREATE TABLE `staff` (
    `name` VARCHAR(12),
    `id_card` CHAR(6)
    `recruit_date` DATE, -- 招募日期
    `last_login` DATETIME
);
```

•***TIMESTAMP***

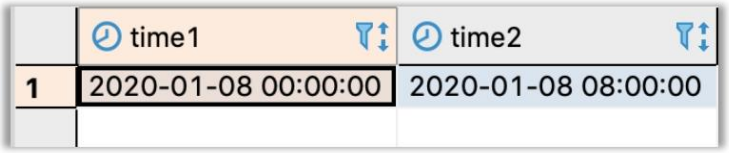
---- Similar to DATETIME, but stores the time as UTC time.
---- When date time is looked up, the stored time will be converted to the time of the current timezone of the client.当查找日期时间时，存储的时间将被转换为 客户端当前时区的时间
---- How to illustrate their difference? Enter the following table with data:

```
create table `ts_test` (
    `time1` timestamp,
    `time2` datetime
);
insert into `ts_test` values('2020-01-08 08:00:00', '2020-01-08 08:00:00');
```

• *Assume your current timezone is UTC+8 (Beijing). Look up the data in ts_test, you will find:*

	time1	time2
1	2020-01-08 08:00:00	2020-01-08 08:00:00

• *Now, in the system, change your timezone to UTC+0 (London) Look up the data in ts_test again, you will now find:*



	time1	time2
1	2020-01-08 00:00:00	2020-01-08 08:00:00

2.6 Column Options

col-name datatype [col-options]

1 ***NOT NULL****: values of this column cannot be null.*
2 ***UNIQUE****: each value must be unique (similar to candidate key on a single attribute)*
3 ***DEFAULT value*** *默认值: Default value for this column if not specified by the user. 在某些数据库管理系统中，如 Microsoft Access，此选项可能无效。*
Example: ***age INT DEFAULT 12*** *如果不设置默认 age 为 12*
4 ***AUTO_INCREMENT*** *在使用 CREATE TABLE 语句创建表时，可以将 AUTO_INCREMENT 应用于键列（主键、唯一键）。它指示数据库管理系统在添加数据时自动插入一个值（通常是 max(col) + 1）。这使得每次插入新行时，自增列都会自动递增。*
---- Must be applied to a key column (primary key/ unique key).
----You are not allowed to use "AUTO_INCREMENT = 2" inside CREATE TABLE
---- ***ALTER TABLE Persons AUTO_INCREMENT = 100;*** *这将使下一次插入数据时，Id 列的起始值为 100*

```
CREATE TABLE Persons (
    id INT UNIQUE NOT NULL AUTO_INCREMENT,
    lastname VARCHAR(255) NOT NULL,
    firstname VARCHAR(255),
    age INT DEFAULT 12,
    city VARCHAR(255)
) AUTO_INCREMENT = 5;
```

• *Implicit Default Values：When a DEFAULT option is not used, the database may give implicit default values depending on the column data type:*
---- If the column can take NULL as a value, the column is defined with an explicit DEFAULT NULL clause.
---- If the column cannot take NULL as a value, MySQL defines the column with no explicit DEFAULT clause.

在数据库中，每个列都有其定义的数据类型和约束条件。如果你没有为一个非 NULL 列提供值，并且也没有定义默认值或自增属性，那么该列将处于未定义的状态，违反了列的定义和约束条件。

为了确保数据的完整性和一致性，数据库管理系统要求你提供一个具体的值或使用其他方式来为该列设置默认值。这可以通过显式提供值、使用 DEFAULT 定义默认值或将列设置为自增（AUTO_INCREMENT）来实现。

```
CREATE TABLE t (
    i INT NOT NULL
);
INSERT INTO t VALUES();
INSERT INTO t VALUES(DEFAULT);
INSERT INTO t VALUES(DEFAULT(i));
```

All statements above gives error in strict mode

INSERT INTO t VALUES(); 这个语句中，你没有提供任何值，也没有使用 DEFAULT 关键字。插入语句要求提供要插入的具体值或使用 DEFAULT 关键字指定默认值，因此这个语句会报错。

INSERT INTO t VALUES(DEFAULT); 这个语句中，你使用了 DEFAULT 关键字，但是 DEFAULT 关键字需要在括号中指定列名，用于指定使用哪个列的默认值。因此，这个语句也会报错。

INSERT INTO t VALUES(DEFAULT(i));这个语句中，你使用了 DEFAULT 关键字，并在括号中指定了列名 i，但是由于表定义中的列 i 没有定义默认值，所以这个语句也会报错。如果你想插入列 i 的默认值，需要确保在表定义中为列 i 指定了默认值，例如：CREATE TABLE t (i INT NOT NULL DEFAULT 0);

3 Tuple Manipulation

CREATE TABLE 语句只定义表的结构，而行数据是通过 INSERT INTO 语句插入到表中的，表的行数取决于你执行的 INSERT INTO 语句的数量

3.1 INSERT

```
INSERT INTO tablename (col1, col2, …)
    VALUES (val1, val2, …),
    :
    (val1,val2,val3);
```

- *The order of column names must be consistent with the order of values.*
- *If you are adding a value to every column, you don't have to list them:*

```
INSERT INTO tablename VALUES (val1, val2, …);
```

Employee

ID	Name	Salary

Employee

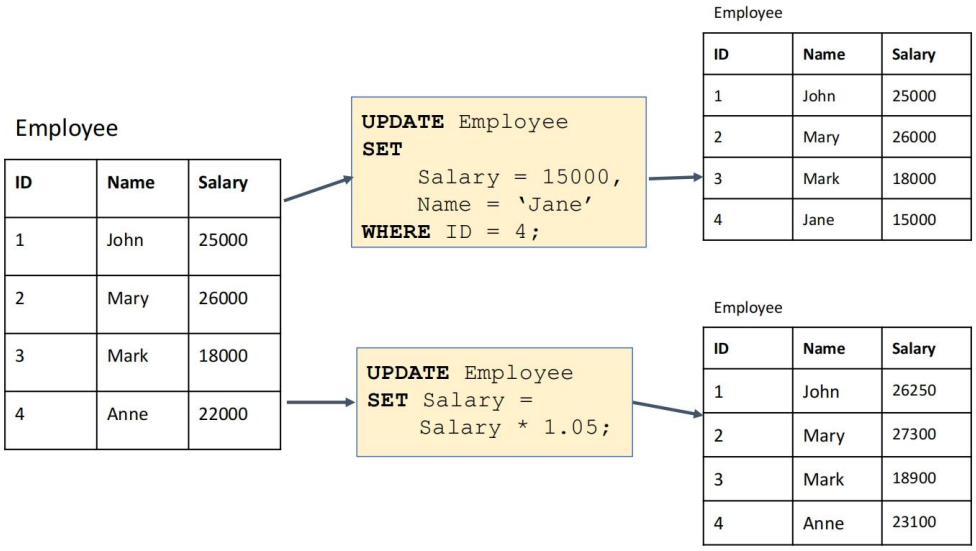
ID	Name	Salary
2	Mary	26000
2	Mary	
2	Mary	26000
3	Max	19000

INSERT INTO `Employee`(`ID`, `Name`, `Salary`)VALUES (2, 'Mary', 26000);
INSERT INTO Employee(Name, ID) VALUES ('Mary', 2);
INSERT INTO Employee VALUES (2, 'Mary', 26000), (3, 'Max', 19000);

3.2 UPDATE

UPDATE table-name SET col1 = val1 [,col2 = val2…][WHERE condition]

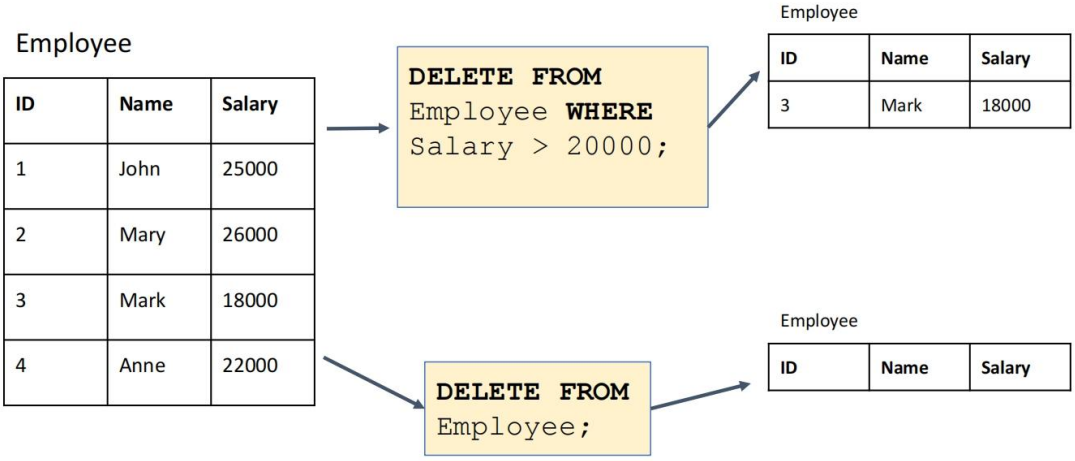
• *All rows where the condition is true have the columns set to the given values. If no condition is given all rows are changed.*
• *Values are constants or can be computed from columns.*



3.3 DELETE

DELETE FROM table-name [WHERE condition]

• *Removes all rows, or those which satisfy a condition. If no condition is given then ALL rows are deleted.*



4 Table Constraints

- *Table constraints can be defined when creating tables.*
- *But you can also add constraints to an existing table.*

```
CREATE TABLE name (
    col-name datatype [col-options],
    :
    col-name datatype [col-options],
    [constraint-1],
    :
    [constraint-n]
);
```

*Syntax of Constraints：**CONSTRAINT name TYPE details;***

- *Constraint name is created so that later this constraint can be removed by referring to its name. If you don't provide a name, one will be generated. Most of them can be defined along with a column or separately*

- *MySQL provides following constraint types*

----PRIMARY KEY
----UNIQUE
----FOREIGN KEY
----INDEX

- *DETAILS 是约束条件的具体细节信息*

*----**PRIMARY KEY (col_name)**，其中 col_name 是指定为主键的列名。*
*----**FOREIGN KEY (col_name) REFERENCES table_name (ref_col_name)**，其中 col_name 是当前表中的列名，table_name 是外键引用的表名，ref_col_name 是外键引用的列名。*

*----**UNIQUE (col_name)**，其中 col_name 是指定为唯一键的列名。*
*----**CHECK (condition)**，其中 condition 是指定的检查条件，可以是一个表达式、函数或子查询等。*

• You can limit the possible values of an attribute by adding a domain constraint.
• A domain constraint can be defined along with the column or separately:

```
CREATE TABLE People (
    id INTEGER PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    sex CHAR NOT NULL CHECK (sex IN ('M','F')),
    CONSTRAINT id_positive CHECK (id > 0)
);
```

• Supported in MySQL 8.0.16+ In earlier versions, these constraints will be ignored.
创建了一个名为 People 的表，其中包含了 id、name 和 sex 三个列。在 sex 列上定义了一个域约束，要求 sex 的取值只能是'M'或'F'，通过 CHECK 子句实现。另外，还定义了一个名为 id_positive 的约束，要求 id 必须大于 0。

4.1 UNIQUE

CONSTRAINT name UNIQUE (col1, col2, …)

• A table can have multiple UNIQUE keys defined.
• The following unique keys are different:
----One unique key (a, b, c): Tuples (1, 2, 3) and (1, 2, 2) are allowed
----Separate unique keys (a) (b) (c):Tuples (1, 2, 3) and (1, 2, 2) are NOT allowed

4.2 Primary Key

CONSTRAINT name PRIMARY KEY (col1, col2 …)

• Constraint name for a primary key is actually ignored by MySQL, but works in other databases.
• PRIMARY KEY also automatically adds UNIQUE and NOT NULL to the relevant column definition

Comparison:
主键（Primary Key）是一种特殊的唯一键，用于标识一张表中的每一行数据。主键的值必须唯一且不为空，不能重复或为 NULL。主键还具有自动递增的功能，即每次插入一行数据时自动加 1，以保证主键的唯一性。
唯一键（Unique Key）也用于标识一张表中的每一行数据，但与主键不同的是，唯一键的值可以为 NULL，且有多个也行。唯一键的值必须唯一，不能重复。

```
CREATE TABLE Branch (
    branchNo CHAR(4),
    street VARCHAR(100),
    city VARCHAR(25),
    postcode VARCHAR(7),
    CONSTRAINT branchUnique UNIQUE (postcode),
    CONSTRAINT branchPK PRIMARY KEY (branchNo)
);
```

Alternative Way:

```
CREATE TABLE Branch2 (
    branchNo CHAR(4) PRIMARY KEY,
    street VARCHAR(100),
    city VARCHAR(25),
    postcode VARCHAR(7) UNIQUE
);
```

The primary key and unique key will automatically be assigned with constraint names.

4.3 Foreign Key

```
CONSTRAINT name
    FOREIGN KEY (col1, col2, ...)
    REFERENCES table-name (col1, col2, ...)     -- 引用其他列表的表名和列
        [ON UPDATE ref_opt ON DELETE ref_opt]
        -- ref_opt: RESTRICT | CASCADE | SET NULL | SET DEFAULT
```

• A foreign key consists of following parts: A constraint name, Columns of the referencing table, Referenced table and referenced columns, Reference options

```
CREATE TABLE branch (
    branchNo CHAR(4) PRIMARY KEY,
    street VARCHAR(50),
    ...
);
CREATE TABLE staff (
    staffNo CHAR(6) PRIMARY KEY,
    fName VARCHAR(20),
    branchNo CHAR(4),
    CONSTRAINT FK_staff_branchNo
        FOREIGN KEY (branchNo) --  必须括号
        REFERENCES branch (branchNo)
);
```

----Foreign key must reference a unique key or primary key
----Data types of both columns must be compatible
----Column list must be enclosed with brackets
*----After applying FK, **`staff`.`branchNo`** (the branchNo column of staff) will be checked by the database, the value must either be: An existing value from `branch`.`branchNo` or null.*

Example 1 ----CODE:DBL1_1

• Given the branch table on the right, insert following tuples into staff

----('S1', 'staff1', 'b001'),
----('S2', 'staff2', 'B007'),
----('S3', 'staff3', 'B001'),
----('S4', 'staff4', 'B002'),
----('S1', 'staff5', 'B002');

Branch

branchNo	street	...
B001	...	...
B002	...	...
B005	...	...

• The black means that it doesn't work true,and -- explain the reason why.
• In MySQL, string comparison is case-insensitive 不区分大小写,so the 'b001' above will violate the primary key constraint. The following code show the solution:

```
CREATE TABLE `branch` (
    `branchNo` CHAR(4) BINARY NOT NULL,
    PRIMARY KEY (`branchNo`),
...);
```

• The BINARY keyword instructs MySQL to compare the characters in the string using their underlying ASCII values rather than just their letters.
----注意是在原表添加 binary
----In other databases, string comparison can be implemented differently!

Foreign Keys and Tuple Updates

• We just saw that attempts to add non-existing branchNo to Staff were rejected by DBMS. But what happens when we change/delete existing branchNo in Branch that are being referenced by Staff?

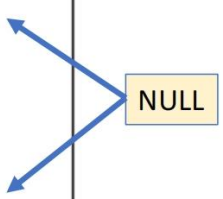
• There are several options when this occurs in Foreign Key Constraint:
*----**RESTRICT:** stop the user from doing it ,which is the default option*
*----**CASCADE:** let the changes flow on*
*----**SET NULL**:make referencing values null*
*----**SET DEFAULT:**make referencing values the default for their column*

• *The four above options can be applied to one or both kinds of the table updates:*
*----**ON DELETE:**What will happen if referenced values are deleted.*
*----**ON UPDATE:**What will happen if referenced values are updated.*

Example

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005



• *Use On Update/Delete Set NULL. Assume we delete 'B005'in Branch table, then all 'B005' in the Staff table will be set to null.But if we change 'B005' to 'B006', all 'B005' will be set to null,which is not a good decision.*

• *Instead Use On Update/Delete Cascade. Assume we change 'B005' to 'B006' in Branch table,then all 'B005' in Staff table will be changed to 'B006',which is reasonable in real world. But if we delete 'B005' in Branch table,then all tuples with 'B005' in Staff table will also be deleted,which is not a good decision.*

• *Instead Use On Update/Delete Set Default. Assume we delete or change 'B005' in Branch table,then all 'B005' in Staff table will be changed to the default value of Staff (branchNo),but this feature is not available in MySQL.*

• *As the execution will stop if use On Update/Delete RESTRICT, so we don't discuss yet.*

Final FK Definition--The default option (restrict) also works

```
CONSTRAINT `FK_staff_branchNo`
    FOREIGN KEY (`branchNo`)
    REFERENCES `branch` (`branchNo`)
        ON DELETE SET NULL
        ON UPDATE CASCADE
```

Example 2 ----CODE:DBL1_2
• *What will happen to the staff table if the following lines were executed?*
1. DELETE FROM branch WHERE branchNo = 'B001';
2. UPDATE branch SET branchNo = 'B007' WHERE branchNo = 'B005';

CONSTRAINT `FK_staff_branchNo`
FOREIGN KEY (`branchNo`)
REFERENCES `branch` (`branchNo`)
ON DELETE SET NULL
ON UPDATE CASCADE

staffNo	fName	branchNo
S1	staff1	b001
S2	staff2	B001
S3	staff3	B002
S4	staff4	B002
S5	staff5	B005

5 Altering Tables

• Add column:
ALTER TABLE table_name
ADD column_name datatype [options like UNIQUE …];

• Drop column:
ALTER TABLE table_name
DROP COLUMN column_name;

• Modify column name and definition:
ALTER TABLE table_name
CHANGE COLUMN col_name new_col_name datatype [col_options];

• Modify column definition only:
ALTER TABLE table_name
MODIFY COLUMN column_name datatype [col_options];

• Add a constraint:
ALTER TABLE table-name
ADD CONSTRAINT name definition;

• Remove a constraint:---Can be used to drop unique keys(index)
ALTER TABLE table-name
DROP INDEX name | DROP FOREIGN KEY name | DROP PRIMARY KEY

• Delete Tables
DROP TABLE [IF EXISTS] table-name1, table-name2…;

• Tables will be dropped in that exact order
----All tuples in the dropped tables will be deleted as well.
• Foreign Key constraints will prevent ***DROPS table (****which is referred by others) to overcome this:*
Method1:Remove the foreign key constraints first then drop the tables.
Method2:Drop the tables in the correct order (eg : a<--b<--c 删除 c,b,a 顺序)
Method3:Turn off foreign key check temporarily *---> go to lab1 for exercise*

DBL1

----CODE: DBL1_0

```
CREATE DATABASE DBL1;
SHOW DATABASES;
USE DBL1;
SHOW TABLES;
CREATE TABLE mytable (
    columnA INT,
    columnB CHAR(11)
);
SHOW TABLES;

CREATE TABLE `staff` (
    `name` VARCHAR(12),
    `is_sick` TINYINT(1),
    `staff_id` INT(11),
    `salary` DECIMAL(5,2)
);
-- TINYINT(1)在MySQL 中的常见用法是用来表示布尔类型或者类似于布尔类型的字段。它可以存储的值只有两种：0 和 1。，TINYINT(0) 是一个无效的数据类型定义。TINYINT 数据类型用于存储范围较小的整数值，可以存储从 -128 到 127（有符号）或从 0 到 255（无符号）之间的整数值。然而，括号中的长度参数用于指定字段的显示宽度，并不影响存储范围。对于 TINYINT 类型，长度参数只接受一个有效的范围值，即 1 到 255。如果长度参数为 0，将被视为无效的定义。因此，TINYINT(0) 在 MySQL 中是不允许的。如果您想创建一个布尔类型或类似于布尔类型的字段，通常使用 TINYINT(1)，其中可以存储 0 和 1 两个值。长度参数 1 表示该字段的显示宽度为 1，但并不影响实际存储和取值范围。

SHOW TABLES;
create table `ts_test` (
    `time1` timestamp,
    `time2` datetime
);
SHOW TABLES;
insert into ts_test values ('2020-01-08 09:00:00', '2020-01-08 08:00:00');
SELECT * FROM ts_test;
SELECT time1, time2 FROM ts_test;
SELECT time1 FROM ts_test;
SELECT time2 FROM ts_test;

-- 以上是输出 table 不同方式，但输出结果只是 01/08/2020，下面这种输出结果是 01/08/2020 09:00:00

SELECT DATE_FORMAT(time1, '%m/%d/%Y %H:%i:%s') AS time1_formatted,
```

```
DATE_FORMAT(time2, '%m/%d/%Y %H:%i:%s') AS time2_formatted FROM ts_test;
create table `student` (
    `id` int unique auto_increment,
    `name` char(12) not null,
    `year` int default 2
) auto_increment = 7;
-- 设置 id 起始值为 7
insert into student (`name`,`year`) values (null, 9);
-- 报错 name 不能 null
insert into student (`year`) values (9);
-- 报错，注意 insert 添加的是一行，name 因为设计 not null 导致初始值不能是默认的 null 导致报错
insert into student (`name`) values ('Jason');
SELECT * FROM student;
-- 结果为 7，Jason，2

create table `student1` (
    `id` int unique ,
    `name` char(12) not null,
    `year` int default 2
) ;
insert into student1 (name) values ('Jason');
SELECT * FROM student1;
-- 结果：  ，Jason,2    没错 id 列空着了

insert into student1 (`name`) values ('Jason1');
SELECT * FROM student1;
-- 这个代码目的是我看到 id unique 于是思考允不允许出现俩个空，事实是可以
--  ，Jason,2
--  ，Jason1,2

create table `student2` (
    `id` int unique auto_increment,
    `name` char(12) not null,
    `year` int default 2
) auto_increment = 7;
INSERT INTO student2 VALUES(12,'ANAS',4);
insert into student2 (`name`) values ('Jason1');
SELECT * FROM student2;
-- 这个代码目的是观察是否在填入后第二个没有设计 id 是从 12+1 还是 7+1，结果是 12+1
-- 结果：12，ANAS，4
--      ：13 , Jason1, 2
DELETE FROM student2;-- 删除表所有内容
```

```
SELECT * FROM student2;
INSERT INTO student2 VALUES(5,'ANAS',4);
insert into student2 (`name`) values ('Jason1');
SELECT * FROM student2;
-- 注意这里先展示结果：
-- 5，ANAS，4
-- 14，Jason1,2
-- 那么这里14是如何来的？13+1  也就是说你删除表内容不会影响你下次递增计数，从12-13-14会一直传下去不会突然变回原来的7，不受删除操作影响

DELETE FROM student2 WHERE name='ANAS';
SELECT * FROM student2;
-- 结果：14，Jason1, 2  注意这里等于号只有一个‘==’会报错

DELETE FROM student2 WHERE name='Jason1';
SELECT * FROM student2;
SHOW TABLES;

create table `student3` (
    `id` int unique auto_increment,
    `name` char(12) not null,
    `year` int default 2
) auto_increment = 7;
INSERT INTO student3 VALUES(5,'ANAS',4);
insert into student3 (`name`) values ('Jason1');
SELECT * FROM student3;
-- 结果：5，ANAS,4
--       :  7, Jason1,2


create table `student4` (
    `id` int unique auto_increment,
    `name` char(12) not null,
    `year` int default 2
) auto_increment = 7;
INSERT INTO student4 VALUES(1,'ANAS',4);
insert into student4 (`name`) values ('Jason1');
SELECT * FROM student4;
-- 3和4是为了验证当输入值小于7时第二个会怎么样
-- 结果：1，ANAS,4
--       :  7, Jason1,2
-- 说明小于7会自动跳到7作为初始值

DROP TABLE student4;
```

```
-- 删除 table student4
SHOW TABLES;

insert into student3 (`name`) values ('Jason2');
SELECT * FROM student3;
-- 的确是从 7 开始加
-- 5, ANAS, 4
-- 7, Jason1, 2
-- 8, Jason2, 2

SELECT * FROM student;
insert into student (`name`) values ('Jason');
SELECT * FROM student;
insert into student (`id`,`name`,`year`) values (234, 'Jason', 3);
SELECT * FROM student;
-- 7,Jason,2
-- 8,Jason,2
-- 234,Jason,3
insert into student (`id`,`name`) values (8, 'Jane');
-- 报错已经有 id 8，id 是 unique

insert into student (`name`) values ('Anna');
SELECT * FROM student;
-- id 最新的是 235


create table a (
    id int,
    name char(2),
    primary key (id)
);

insert into a (`name`) values ('go');
-- 报错，主键的值必须唯一且不为空，不能重复或为 NULL。主键还具有自动递增的功能，
insert into a (id,`name`) values (112,'go');
SELECT * FROM a;

insert into a (`name`) values ('go');
-- 报错，原因：虽然主键还具有自动递增的功能，但是这个需要设置为自动递增，也就是说你需要在代码中写入： id int auto_increment,

DROP DATABASE DBL1;
SHOW DATABASES;
```

```
CREATE DATABASE DBL1;
SHOW DATABASES;
USE DBL1;

CREATE TABLE student(
    id int ,
    name VARCHAR(255),
    mark int,
    CONSTRAINT positive_mark check(mark>0),
    CONSTRAINT pri_id PRIMARY KEY(id) --  非常重要，reference key  必要条件
);
INSERT INTO student VALUES(1,'Amy',-12);
INSERT INTO student VALUES(1,'Amy',12);
INSERT INTO student(id) VALUES(2);
SELECT*from student;

CREATE TABLE performance(
    id int ,
    office varchar(23),
    CONSTRAINT refer_id
        FOREIGN KEY (id)
        references student(id)
            ON DELETE SET NULL
            ON UPDATE CASCADE
);
INSERT INTO performance values(12,'as');
INSERT INTO performance values(1,'as');
SELECT*from performance;

DELETE from student WHERE id=1;
SELECT*from student;
-- 2,null,null
SELECT*from performance;
-- null,'as'

INSERT INTO performance values(2,'do');
INSERT INTO student VALUES(3,'dd',23);
UPDATE student set id=4 where id =2;
SELECT*from student;
-- 3,'dd',23
-- 4,null,null
SELECT*from performance;
-- null,as
```

```
-- 4,do   把原来的 2 变为 4
UPDATE performance set id=2 where id =4; --  报错原因只能在父表做删改，在子表和限制冲突
DROP DATABASE dbl1;
SHOW DATABASES;

--Example 1 ----CODE:DBL1_1

SHOW DATABASES;
CREATE DATABASE DBL1_1;
SHOW DATABASES;
USE dbl1_1;

CREATE TABLE `branch` (
  `branchNo` char(4) NOT NULL,
  `street` varchar(50) DEFAULT NULL,
  `city` varchar(20) DEFAULT NULL,
  `postcode` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`branchNo`)
);
CREATE TABLE `staff` (
  `staffNo` char(6) NOT NULL,
  `fName` varchar(20) DEFAULT NULL,
  `branchNo` char(4) DEFAULT NULL,
  PRIMARY KEY (`staffNo`),
  CONSTRAINT `FK_staff_branchNo` FOREIGN KEY (`branchNo`) REFERENCES `branch` (`branchNo`)
);
insert into branch values ('B001', 'city1', 'street1', 'postcode1'), ('B002', 'city2', 'street2', 'postcode2'), ('B005', 'city5', 'street5', 'postcode5');
SELECT* FROM branch;

insert into staff values ('S1', 'staff1', 'b001');   -- insert into staff values ('S1', 'staff1', 'b001');语句时，并不会触发外键约束。虽然'b001'与'B001'在大小写上有区别，但对于 MySQL 数据库来说，它们被视为相同的值。
insert into staff values ('S2', 'staff2', 'B007');
-- 报错，B007 没有存储
insert into staff values ('S3', 'staff3', 'B001');
-- 成功
insert into staff values ('S4', 'staff4', 'B002');
-- 成功
insert into staff values ('S1', 'staff5', 'B002');
-- 报错，'S1'已经存储了，这是主键不能重复
insert into staff values ('S2', 'staff5', 'b002');
```

```
SELECT* FROM STAFF;
-- 'S1', 'staff1', 'b001'
-- S2', 'staff5', 'b002'
-- 'S3', 'staff3', 'B001'
-- 'S4', 'staff4', 'B002'

CREATE TABLE `branch_binary` (
  `branchNo` char(4) binary NOT NULL,
  `street` varchar(50) DEFAULT NULL,
  `city` varchar(20) DEFAULT NULL,
  `postcode` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`branchNo`)
);
CREATE TABLE `staff_binary` (
  `staffNo` char(6) NOT NULL,
  `fName` varchar(20) DEFAULT NULL,
  `branchNo` char(4) BINARY DEFAULT NULL,
  PRIMARY KEY (`staffNo`),
  CONSTRAINT `fk`
       FOREIGN KEY (`branchNo`)
       REFERENCES `branch_binary`(`branchNo`)
            ON DELETE RESTRICT
            ON UPDATE RESTRICT
);
insert into `branch_binary` values ('B001', 'city1', 'street1', 'postcode1');

Select * from branch_binary;
insert into `staff_binary` values ('s001', 'city1x', 'b001');   -- 报错：'b001'不能小写
insert into `staff_binary` values ('s001', 'city1x', 'B001');   -- 成功
DROP DATABASE dbl1_1;
SHOW DATABASES;

--Example 2 ----CODE:DBL2_2

SHOW DATABASES;
CREATE DATABASE DBL1_2;
SHOW DATABASES;
USE dbl1_2;
CREATE TABLE `branch` (
  `branchNo` char(4) NOT NULL,
  `street` varchar(50) DEFAULT NULL,
  `city` varchar(20) DEFAULT NULL,
  `postcode` varchar(10) DEFAULT NULL,
  PRIMARY KEY (`branchNo`)
```

```
);
CREATE TABLE `staff` (
  `staffNo` char(6) NOT NULL,
  `fName` varchar(20) DEFAULT NULL,
  `branchNo` char(4) DEFAULT NULL,
  PRIMARY KEY (`staffNo`),
  CONSTRAINT `FK_staff_branchNo`
      FOREIGN KEY (`branchNo`)
      REFERENCES `branch` (`branchNo`)
          ON DELETE SET NULL
          ON UPDATE CASCADE
);

insert into branch values
('B001', 'city1', 'street1', 'postcode1'),
('B002', 'city2', 'street2', 'postcode2'),
('B005', 'city5', 'street5', 'postcode5');

insert into staff values
('S1', 'staff1', 'b001'),
('S2', 'staff2', 'B001'),
('S3', 'staff3', 'B002'),
('S4', 'staff4', 'B002'),
('S5', 'staff5', 'B005');

SELECT * from branch;
SELECT * from staff;


delete from branch where branchNo = 'B001';
update branch set branchNo = 'B007' where branchNo = 'B005';
SELECT * from branch;
SELECT * from staff;
-- ('S1', 'staff1', null),
-- ('S2', 'staff2', null),
-- ('S3', 'staff3', 'B002'),
-- ('S4', 'staff4', 'B002'),
-- ('S5', 'staff5', 'B007');

alter table staff add `lName` varchar(20) not null;
SELECT * from staff;-- 此时不会报错，原因现在没有 insert 只会改变老的结构
-- ('S1', 'staff1', null，  ),
-- ('S2', 'staff2', null，  ),
-- ('S3', 'staff3', 'B002'，  ),
```

```
-- ('S4', 'staff4', 'B002',    ),
-- ('S5', 'staff5', 'B007',    );
alter table staff drop column `lName`;
alter table staff change column `fName` `first_name` varchar(20) not null;
alter table staff modify column `first_name` varchar(40) not null;
SELECT * from staff;
SELECT * from branch;

alter table branch add constraint ck_branch unique (street);
alter table staff add constraint fk_staff_staff
      foreign key (branchNo) references branch (branchNo);
alter table staff drop primary key;
alter table staff drop foreign key fk_staff_staff;
alter table branch drop index ck_branch;

drop table branch; -- will fail, because of foreign key.
SELECT * from branch;
alter table staff drop foreign key FK_staff_branchNo;
drop table branch; -- will succeed now!

Drop DATABASE dbl1_2;
show DATABASES;



SHOW DATABASES;
CREATE DATABASE DBL1_2;
SHOW DATABASES;
USE dbl1_2;


CREATE TABLE c (
   c1 int PRIMARY KEY
);

CREATE TABLE b (
   b1 int PRIMARY key,
   b2 int,
   CONSTRAINT b2_c FOREIGN KEY(b2) REFERENCES c(c1)
         on DELETE CASCADE
         ON UPDATE SET NULL
);

CREATE TABLE a (
```

```
    a1 int PRIMARY key,
    a2 int,
    CONSTRAINT a2_b1 FOREIGN KEY(a2) REFERENCES b(b1)
          on UPDATE CASCADE
          ON DELETe set NULL
);


insert into c values
(31),(32),(33);


insert into b values
(21,31),(22,32),(23,32),(24,33);

insert into a values
(11,21),(12,22),(13,23);

SELECT * from c;
SELECT * from b;
SELECT * from a;


update c set c1 = 34 where c1 = 32;
SELECT * from c;
SELECT * from b;
SELECT * from a;


update b set b1 = 25 where b2 >= 32;
delete from c where c1 = 33;
delete from b where b1 = 21;


SELECT * from c;
SELECT * from b;
SELECT * from a;
drop DATABASE dbl1_2;
```

2 SQL:Select

SELECT [DISTINCT | ALL]
column-list FROM table-names
[WHERE condition]
[ORDER BY column-list]
[GROUP BY column-list]
[HAVING condition]

1 BASIC SYNTAX

SELECT col1[,col2…] FROM table-name;

	id	code	mark
1	S103	DBS	72
2	S103	IAI	58
3	S104	PR1	68
4	S104	IAI	65
5	S106	PR2	
6	S107	PR1	
7	S107	PR2	
8	S107	IAI	

```
select id, code
    from grade;
```

	id	code
1	S103	DBS
2	S103	IAI
3	S104	PR1
4	S104	IAI
5	S106	PR2
6	S107	PR1
7	S107	PR2
8	S107	IAI

2 DISTINCT and ALL

- *Using **DISTINCT** after the **SELECT** keyword removes duplicates .*
- *Using **ALL** retains duplicates,and **ALL** is used as a default condition if neither is supplied ,so you can get duplicates without add anything.*
- *These will work over multiple columns in future.*

```
SELECT ALL Last
   FROM Student;
```

Last
Smith
Jones
Brown
Jones
Brown

```
SELECT DISTINCT Last
   FROM Student;
```

Last
Smith
Jones
Brown

3 AS--give a column a new name

select a, b, a+b as sum from dup_test;

	a	b	sum
1	1	1	2
2	2	1	3
3	1	2	3
4	1	1	2

4 Where clause : https://dev.mysql.com/doc/refman/8.0/en/expressions.html

SELECT * FROM table-name WHERE predicate;

• ***WHERE*** *clause restricts rows that are returned according to Predicate.*

---- Predicate can be understood as an expression that either returns a true or false (for numbers, non-zero or zero) and only rows that satisfy the condition will appear in the final Result.

*---- * means getting all columns of that table.*

----The comparison of date and time can be done just like numbers.

SELECT * FROM table-name WHERE date-of-event < '2012-01-01';

But you can also search for dates like a string:

SELECT * FROM table-name WHERE date-of-event LIKE '2014-11-%';

---- You can not use aggregate functions in Where

Expression	Meaning
Mark < 40	The value of column `mark` is less than 40
First = 'John'	The value of column `First` equals to 'John'
First = Last	`First` equals to `Last`
First IS NULL	`First` column has no value
First != 'John' First <> 'John'	The value of column `First` NOT equals to 'John'
(First = 'John') AND (Last = 'Smith')	`First` equals to 'John' and `Last` equals to 'Smith'
(Mark < 40) OR (Mark > 70)	Mark is lower than 40 or higher than 70

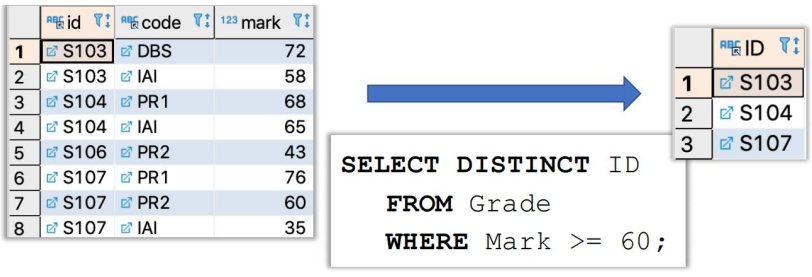
---- select * from grade where mark - 72;这里选择 mark-72 不为 0 的行

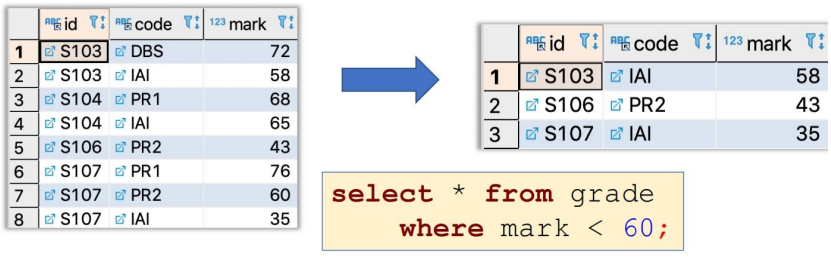
• *The Operation process of where:*

---- Get the table (the FROM part)

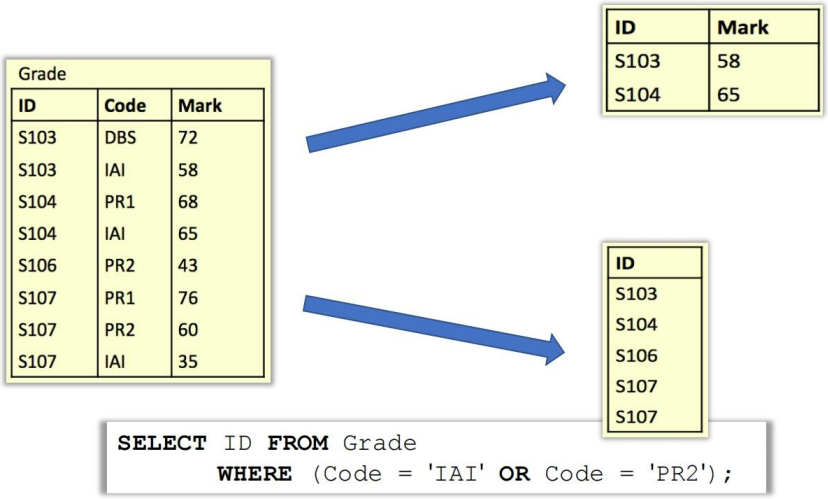
---- For each tuple, assess the WHERE clause:True -> accepted/ False-> removed

---- Get columns (the SELECT part)





```
SELECT ID, Mark FROM Grade
       WHERE (Code = 'IAI') AND (Mark >= 50);
```




```
SELECT ID FROM Grade
       WHERE (Code = 'IAI' OR Code = 'PR2');
```

• Word search is commonly used for searching product catalogue, which might need to use partial keywords

---- For example: given a database of books, searching for "crypt" might return : "Cryptonomicon" by Neil Stephenson ,"Applied Cryptography" by Bruce Schneier.

*---- We can use the **LIKE** keyword to execute string comparisons in queries. Like is not the same as '=' because it allows wildcard characters. It is NOT normally case sensitive 它通常不区分大小写*

4.1 LIKE

• The '%' character can represent any number of characters including none.

SELECT * FROM books WHERE bookName LIKE 'crypt%';

---- This will return "Cryptography Engineering" and "Cryptonomicon" but not "Applied Cryptography"(不是以 crpt 开头，要获得需要改成 '%crypt%')

• The '_' character represents exactly one character, not including none

SELECT * FROM books WHERE bookName LIKE 'cloud_';

---- This will return "Clouds" but not "Cloud" or "cloud computing

*• Sometimes you might need to search for a set of words .In order to find specific words, you can link them with **AND.***

```
SELECT * FROM books
    WHERE bookName LIKE '%crypt%' AND bookName LIKE '%cloud%';
```

*• Sometimes you might need to search for a set of words .In order to find arbitrary(任意的) eligible(符合条件的) words, you can link them with **OR.***

```
SELECT * FROM books WHERE
    bookName LIKE '%crypt%' OR bookName LIKE '%cloud%';
```

Example--Find track title containing either the string 'boy' or 'girl'

Track

cdID	Num	Track_title	Time	aID
1	1	Violent	239	1
1	2	Every Girl	410	1
1	3	Breather	217	1
1	4	Part of Me	279	1
2	1	Star	362	1
2	2	Teaboy	417	2

CD

cdID	Title	Price
1	Mix	9.99
2	Compilation	12.99

Artist

aID	Name
1	Stellar
2	Cloudboy

```
SELECT Track_title FROM Track
    WHERE Track_title LIKE '%boy%' OR Track_title LIKE '%girl%';
```

4.2 The following statements will return Boolean values in the SELECT section:

```
CREATE DATABASE ex1;
use ex1;
create TABLE place(
      pastcode varchar(255)
);
insert into place VALUES('sdf'),('gbsss'),('gbn');
SELECT pastcode LIKE 'gb%' FROM place;
-- 0
-- 1
-- 1
create table staff(
      id int
);
insert into staff VALUES(1),(3),(8),(5);
SELECT id BETWEEN 1 AND 5 FROM staff;
-- 1
-- 1
-- 0
--1
DROP DATABASE EX1;
SHOW DATABASES;
```

5 Select with Cartesian Product

Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	Brown
S106	Mark	Jones
S107	John	Brown

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

Course

Code	Title
DBS	Database Systems
PR1	Programming 1
PR2	Programming 2
IAI	Introduction to AI

- *Cartesian product of two tables can be obtained by using: (5*8 rows)*

  SELECT * FROM Student, Grade;

- *If the tables have columns with the same name, we resolve this condition by referencing columns with the table name. For example:*

  SELECT Student.ID FROM Student, Grade WHERE Student.ID = Grade.ID;

- *The following statement below is wrong as the ID is ambiguous 模糊不清的*

  ~~***SELECT ID FROM Student, Grade WHERE Student.ID = Grade.ID;***~~

SELECT First, Last, Mark FROM Student, Grade
WHERE (Student.ID = Grade.ID) AND (Mark >= 40);

Student

ID	First	Last
S103	John	Smith
S104	Mary	Jones
S105	Jane	Brown
S106	Mark	Jones
S107	John	Brown

Grade

ID	Code	Mark
S103	DBS	72
S103	IAI	58
S104	PR1	68
S104	IAI	65
S106	PR2	43
S107	PR1	76
S107	PR2	60
S107	IAI	35

First	Last	Mark
John	Smith	72
John	Smith	58
Mary	Jones	68
Mary	Jones	65
Mark	Jones	43
John	Brown	76
John	Brown	60

- *SELECT can use in more than 2 tables:* ***SELECT * FROM Student, Grade, Course WHERE Student.ID = Grade.ID AND Course.Code = Grade.Code AND Mark >= 40;***

Student Grade Course

ID	First	Last	ID	Code	Mark	Code	Title
S103	John	Smith	S103	DBS	72	DBS	Database Systems
S103	John	Smith	S103	IAI	58	IAI	Introduction to AI
S104	Mary	Jones	S104	PR1	68	PR1	Programming 1
S104	Mary	Jones	S104	IAI	65	IAI	Introduction to AI
S106	Mark	Jones	S106	PR2	43	PR2	Programming 2
S107	John	Brown	S107	PR1	76	PR1	Programming 1
S107	John	Brown	S107	PR2	60	PR2	Programming 2

Student.ID = Grade.ID Grade.Code = Course.Code

```
Create DATABASE ex1;
use ex1;
drop table if exists grade, student, course;
create table student (
	id char(4) primary key,
	first varchar(20) not null,
	last varchar(20) not null
);
insert into student values
('S103','John','Smith'),
('S104','Mary','Jones'),
('S105','Jane','Brown'),
('S106','Mark','Jones'),
('S107','John','Brown');
create table course (
	code char(3) primary key,
	title varchar(20) not null
);
insert into course values
('DBS','Database Systems'),
('PR1','Programming 1'),
('PR2','Programming 2'),
('IAI','Introduction to AI');
create table grade (
	id char(4),
	code char(3),
	mark int,
	constraint fk_grade_student foreign key
		(id) references student(id),
	constraint fk_grade_code foreign key
		(code) references course(code)
);
insert into grade values
('S103','DBS',72),('S103','IAI',58),
('S104','PR1',68),('S104','IAI',65),
('S106','PR2',43),('S107','PR1',76),
('S107','PR2',60),('S107','IAI',35);
SELECT * from student;
select * from grade;
select * from course;
SELECT * FROM Student, Grade, Course
WHERE Student.ID = Grade.ID AND Course.Code = Grade.Code AND Mark >= 40;
DROP DATABASE ex1;
```

6 AS

SELECT column [as] ... from Table [as] .... where.... ~~as~~

---- as is optional in syntax like: col1 t1, col2 t2 is also true

---- You cannot use renaming in a WHERE clause

SELECT E.ID AS empID, E.Name, W.Department FROM Employee E, WorksIn W
WHERE E.ID = W.ID;

Employee

ID	Name
123	John
124	Mary

WorksIn

ID	Department
123	Marketing
124	Sales
124	Marketing

empID	Name	Department
123	John	Marketing
124	Mary	Sales
124	Mary	Marketing

6.1 Self-join using As

SELECT A.Name FROM Employee A, Employee B
WHERE A.Dept = B.Dept AND B.Name = 'Andy';

Employee

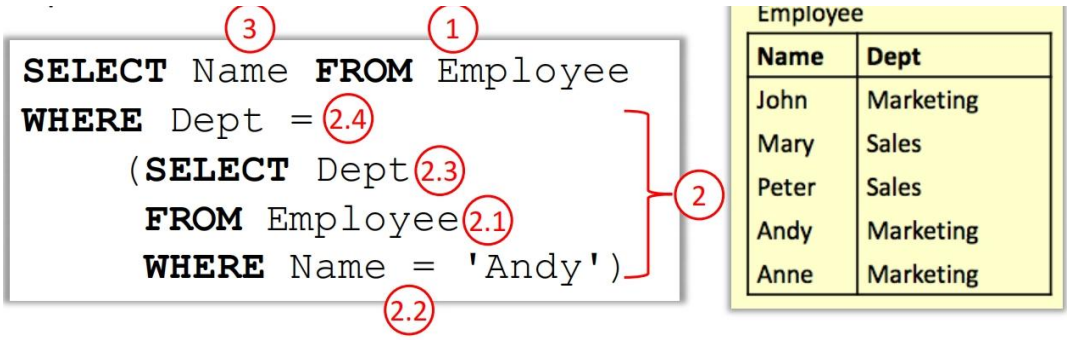
Name	Dept
John	Marketing
Mary	Sales
Peter	Sales
Andy	Marketing
Anne	Marketing

A.Name
John
Andy
Anne

7 Subquery

*• A **SELECT** statement can be nested 嵌套 inside another query to be a subquery . The results of the subquery are passed back to the location of it.*

• The Operation process of all :
---- Get the table (the FROM outside part)
----The WHERE clause outside part
---- Get the table (the FROM inside part)
----The WHERE clause inside part
---- Get columns (the SELECT inside part)
---- Continue to set true/false in The WHERE clause outside part
---- Get columns (the SELECT outside part)



• *You can use a subquery between* ***FROM*** *and* ***WHERE.*** *But the result must be renamed if it is used as a entity.*

SELECT * FROM (SELECT name, email FROM teachers) AS t
WHERE t.email IS NOT NULL;

• *A subquery may return a set of values rather than a single value,and we cannot directly compare a single value to a set. Therefore, we need some syntax to handle comparison between values and sets:*
---- IN : checks to see if a value is in a set
---- EXISTS : checks to see if a set is empty
---- ALL/ANY : checks to see if a relationship between every member of a set and specific value
---- NOT : can be used with any of the above 4

7.1 In

• *Using IN we can see if a given value is in a set of values*
SELECT columns FROM tables
WHERE col IN set; -- 这里 set 指竖直下 column 属性的集合
• *NOT IN checks to see if a given value is not in the set*
SELECT columns FROM tables
WHERE col NOT IN set;
• *The set can be given explicitly or can be produced in a Subquery*
SELECT id FROM student
WHERE id IN ('S103', 'S104');

Example:

SELECT * FROM Employee
WHERE Department IN ('Marketing','Sales');
<=> SELECT * FROM Employee
WHERE Department = 'Marketing' OR Department = 'Sales';

Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane

SELECT * FROM Employee
WHERE Name NOT IN (SELECT Manager FROM Employee);

Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Peter	Sales	Jane

7.2 EXISTS

- *Using **EXISTS** we can see whether there is at least one element in a given set*
  SELECT columns FROM tables
  WHERE EXISTS set;
- ***NOT EXISTS** is true if the set is empty*
  SELECT columns FROM tables
  WHERE NOT EXISTS set;

Example:

SELECT * FROM Employee AS E1
WHERE EXISTS (SELECT * FROM Employee AS E2
WHERE E1.Name = E2.Manager);

Employee

Name	Department	Manager
John	Marketing	Chris
Mary	Marketing	Chris
Chris	Marketing	Jane
Peter	Sales	Jane
Jane	Management	

Name	Dept	Manager
Chris	Marketing	Jane
Jane	Management	

如何理解 ： E1.Name = E2.Manager 显然不可能一捆和一捆比较，这里是每一行的特定值比较，所以是取出 e1 的每一行的值先放到 E1.Name 上

1 行 John --> 没有一致的，这里 SELECT * FROM Employee AS E2 WHERE E1.Name = E2.Manager 就会返回 none ---> e1 第 1 行就不取

2 行 Mary--> 没有一致的，这里 SELECT * FROM Employee AS E2 WHERE E1.Name = E2.Manager 就会返回 none ---> e1 第 2 行就不取

3 行 Chris--> 有一致的，这里 SELECT * FROM Employee AS E2 WHERE E1.Name = E2.Manager 就会返回(('John','Marketing','Chris'),('Mary','Marketing','Chris')) ---> e1 第 3 行就取

4 行 Peter--> 没有一致的，这里 SELECT * FROM Employee AS E2 WHERE E1.Name = E2.Manager 就会返回 none ---> e1 第 4 行就不取

5 行 Jane--> 有一致的，这里 SELECT * FROM Employee AS E2 WHERE E1.Name = E2.Manager 就会返回(('Chris','Marketing','Jane'),('Peter','Sales','Jane')) ---> e1 第 5 行就取

```
CREATE DATABASE ex1;
use ex1;
create TABLE mytable(
        Name varchar(255),
        Department varchar(255),
        Manager varchar(255)
);

INSERT INTO mytable values
('John','Marketing','Chris'),
('Mary','Marketing','Chris'),
('Chris','Marketing','Jane'),
('Peter','Sales','Jane'),
('Jane','Management',null);
SELECT * FROM mytable AS E1 WHERE EXISTS (SELECT * FROM mytable AS E2 WHERE E1.Name = E2.Manager);

DROP DATABASE ex1;
```

7.3 ALL & ANY -- compare a single value to a set of values. *They are used with comparison operators like = , >, <, <>, >=, <= .*

Comparison:

---- val = ANY (set) goes true if there is at least one member of the set equal to value

---- val = ALL (set) goes true if all members of the set are equal to the value

Example:

Name	Salary
Mary	20,000
John	15,000
Jane	25,000
Paul	30,000

SELECT Name FROM Employee
WHERE Salary >= ALL (SELECT Salary FROM Employee);

Name
Paul

SELECT Name FROM Employee
WHERE Salary >ANY (SELECT Salary FROM Employee);

Name
Mary
Jane
Paul

8 Joins

8.1 CROSS JOIN: same as Cartesian product

SELECT * FROM A, B; = SELECT * FROM A CROSS JOIN B;

8.2 INNER JOIN:
*---- same as Cartesian product and Cross Join in many ways, except **Using***

Example:

```
CREATE DATABASE ex1;
use ex1;
create table staff(
    id    int,
    office_loc varchar(255)
);
create table teacher(
    id    int,
    moudle varchar(255),
    name varchar(255)
);
insert into staff VALUES
(1,'SD-11'),
(2,'SA-21'),
(3,'SB-31');
```

insert into teacher VALUES
(1,'cpt111','Peter'),
(2,'cpt107','Amy'),
(3,'cpt103','Janne');

SELECT * from staff,teacher;
<=> SELECT * from staff cross join teacher;
<=> SELECT * from staff inner join teacher;

id	office_loc	id	moudle	name
3	SB-31	1	cpt111	Peter
2	SA-21	1	cpt111	Peter
1	SD-11	1	cpt111	Peter
3	SB-31	2	cpt107	Amy
2	SA-21	2	cpt107	Amy
1	SD-11	2	cpt107	Amy
3	SB-31	3	cpt103	Janne
2	SA-21	3	cpt103	Janne
1	SD-11	3	cpt103	Janne

SELECT * from staff cross join teacher where staff.id=teacher.id;
<=> SELECT * from staff cross join teacher on staff.id=teacher.id;
<=> SELECT * from staff inner join teacher where staff.id=teacher.id;
<=> SELECT * from staff inner join teacher on staff.id=teacher.id;

id	office_loc	id	moudle	name
1	SD-11	1	cpt111	Peter
2	SA-21	2	cpt107	Amy
3	SB-31	3	cpt103	Janne

******* SELECT * from staff inner join teacher using (id);***

id	office_loc	moudle	name
1	SD-11	cpt111	Peter
2	SA-21	cpt107	Amy
3	SB-31	cpt103	Janne

注意这里 id 列只有一个，代表在 teacher.id=staff.id 时对应的笛卡尔乘积列表

8.3 NATURAL JOIN

In 8.2 example:

SELECT * from staff inner join teacher using (id);
<=> SELECT * from staff natural join teacher;

id	office_loc	moudle	name
1	SD-11	cpt111	Peter
2	SA-21	cpt107	Amy
3	SB-31	cpt103	Janne

Another example:

```
CREATE DATABASE ex1;
use ex1;
create table staff(
    id    int,
    office_loc varchar(255),
    salary int
);
create table teacher(
    id    int,
    moudle varchar(255),
    name varchar(255),
    salary varchar(255)
);
insert into staff VALUES
(1,'SD-11',1000),
(2,'SA-21',2000),
(3,'SB-31',3000);
insert into teacher VALUES
(1,'cpt111','Peter','1000'),
(2,'cpt107','Amy','1500'),
(3,'cpt103','Janne','1200');
```

SELECT * from staff inner join teacher using (id);

id	office_loc	salary	moudle	name	salary
1	SD-11	1000	cpt111	Peter	1000
2	SA-21	2000	cpt107	Amy	1500
3	SB-31	3000	cpt103	Janne	1200

*SELECT * from staff inner join teacher using (salary);*

salary	id	office_loc	id	moudle	name
1000	1	SD-11	1	cpt111	Peter

Update teacher set salary ='2000' WHERE id=2;
*SELECT * from staff inner join teacher using (salary);*

salary	id	office_loc	id	moudle	name
1000	1	SD-11	1	cpt111	Peter
2000	2	SA-21	2	cpt107	Amy

Update teacher set salary ='1000' WHERE id=2;
*SELECT * from staff inner join teacher using (salary);*

salary	id	office_loc	id	moudle	name
1000	1	SD-11	1	cpt111	Peter
1000	1	SD-11	2	cpt107	Amy

*SELECT * from staff inner join teacher using (id,salary);*
*<=> SELECT * from staff NATURAL join teacher;*

id	salary	office_loc	moudle	name
1	1000	SD-11	cpt111	Peter

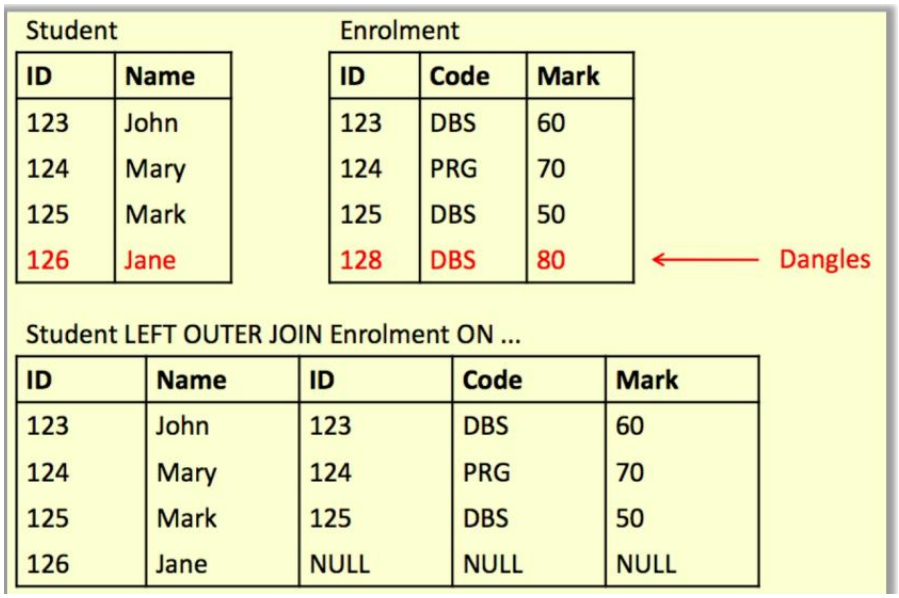
----在表格中，使用 inner join 和 natural join 在属性名称相同数据类型不同时，数字和字符串的值相等，可以被获取到
----Using 只是针对括号内相同属性的相同值，对于相同属性的不同值不会获取
----Natural join 是获取针对俩个表格合并所有属性相同下值相同的表格，对于相同属性的不同值不会获取，等价于 inner join ...using(二者所有属性相同的属性）

8.4 OUTER JOIN

SELECT col FROM table1 LEFT OUTER JOIN table2 ON condition;
SELECT col FROM table1 RIGHT OUTER JOIN table2 ON condition;
SELECT col FROM table1 FULL OUTER JOIN table2 ON condition;

Example:

SELECT * FROM Student LEFT OUTER JOIN Enrolment ON Student.ID = Enrolment.ID;



Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

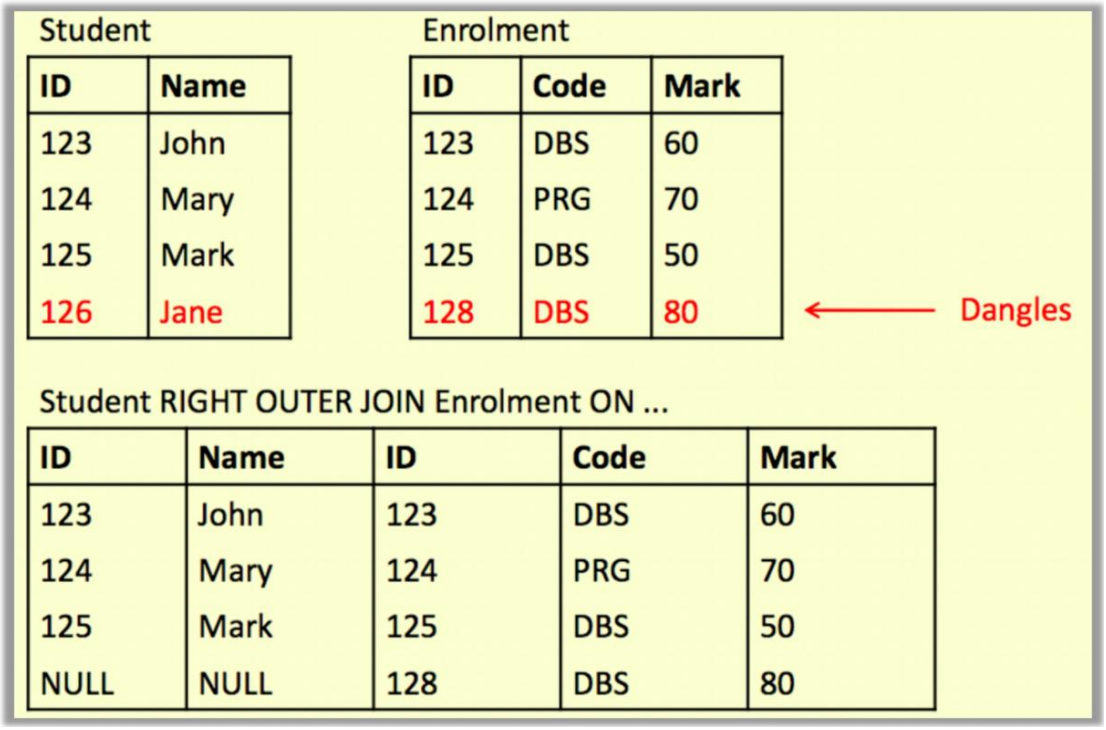
← Dangles

Student LEFT OUTER JOIN Enrolment ON ...

ID	Name	ID	Code	Mark
123	John	123	DBS	60
124	Mary	124	PRG	70
125	Mark	125	DBS	50
126	Jane	NULL	NULL	NULL

注意这里表 student 先出现所以为左边，left outer join 将左边视为主键意思是左边每一行一定都在最终出现，没有符合条件的就输出 null 一行，有就不会输出 null 这一行。左边的每一行会右边的每一行比较如果有多个符合则会有多行左边一致右边为对应的一行

SELECT * FROM Student Right OUTER JOIN Enrolment ON Student.ID = Enrolment.ID;



Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code	Mark
123	DBS	60
124	PRG	70
125	DBS	50
128	DBS	80

← Dangles

Student RIGHT OUTER JOIN Enrolment ON ...

ID	Name	ID	Code	Mark
123	John	123	DBS	60
124	Mary	124	PRG	70
125	Mark	125	DBS	50
NULL	NULL	128	DBS	80

• Only Left and Right outer joins are supported in MySQL. If you really want a FULL outer join: ***SELECT col FROM table1 FULL OUTER JOIN table2 ON condition; <=> (SELECT col FROM table1 LEFT OUTER JOIN table2 ON condition;) UNION (SELECT col FROM table1 RIGHT OUTER JOIN table2 ON condition);***

• Why Using Outer Joins? Sometimes an outer join is the most practical approach. We may encounter NULL values, but may still wish to see the existing information.

Example:
For students graduating in absentia, find a list of all student IDs, names, addresses, phone numbers and their final degree classifications.

Student

ID	Name	aID	pID	Grad
123	John	12	22	C
124	Mary	23	90	A
125	Mark	19	NULL	A
126	Jane	14	17	C
127	Sam	NULL	101	A

Phone

pID	pNumber	pMobile
17	1111111	07856232411
22	2222222	07843223421
90	3333333	07155338654
101	4444444	07213559864

Degree

ID	Classification
123	1
124	2:1
125	2:2
126	2:1
127	3

Address

aID	aStreet	aTown	aPostcode
12	5 Arnold Close	Nottingham	NG12 1DD
14	17 Derby Road	Nottingham	NG7 4FG
19	1 Main Street	Derby	DE1 5FS
23	7 Holly Avenue	Nottingham	NG6 7AR

SELECT ID, Name, aStreet, aTown, aPostcode,pNumber, Classification FROM((Student LEFT OUTER JOIN Phone ON Student.pID = Phone.pID)LEFT OUTER JOIN Address ON Student.aID = Address.aID) INNER JOIN Degree ON Student.ID = Degree.ID WHERE Grad = 'A';

- An Inner Join with Student and Address will ignore Student 127, who doesn't have an address record
- An Inner Join with Student and Phone will ignore student 125, who doesn't have a phone record

Student

ID	Name	aID	pID	Grad
123	John	12	22	C
124	Mary	23	90	A
125	Mark	19	NULL	A
126	Jane	14	17	C
127	Sam	NULL	101	A

Address

aID	aStreet	aTown	aPostcode
12	5 Arnold Close	Nottingham	NG12 1DD
14	17 Derby Road	Nottingham	NG7 4FG
19	1 Main Street	Derby	DE1 5FS
23	7 Holly Avenue	Nottingham	NG6 7AR

Phone

pID	pNumber	pMobile
17	1111111	07856232411
22	2222222	07843223421
90	3333333	07155338654
101	4444444	07213559864

9 Order By

SELECT columns FROM tables
WHERE condition
ORDER BY cols [ASC | DESC]

---- The ORDER BY clause sorts the results of a query. You can sort in ascending or descending order. Multiple columns can be given. You should not choose to order by a column that is not in the result.But the following is possible :SELECT y / 100 AS y2 FROM a ORDER BY y DESC;
---- ORDER BY COL is also true,generate the column in ascending

```
CREATE database ex1;
use ex1;
CREATE table mytable(
        x int,
        y int
);
insert into mytable VALUES
(1,12),
(2,12),
(3,12),
(4,12);
SELECT x/ 100 AS x2 FROM mytable ORDER BY x DESC;
Drop TABLE mytable;

CREATE table mytable(
        x int,
        y int
);
insert into mytable VALUES
(1,20),
(2,12),
(3,4),
(4,-4);

      SELECT x/ 100 AS x2 FROM mytable ORDER BY x ASC;
<=> SELECT x/ 100 AS x2 FROM mytable ORDER BY x2 ASC;
<=> SELECT x/ 100 AS x2 FROM mytable ORDER BY x2 ;
<=> SELECT x/ 100 AS x2 FROM mytable ORDER BY y DESC;
```

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
James	PR1	43
James	PR2	35
Jane	IAI	54

```
SELECT * FROM Grades
   ORDER BY
   Code ASC, Mark DESC;
```

Name	Code	Mark
Mary	DBS	60
John	DBS	56
John	IAI	72
Jane	IAI	54
James	PR1	43
James	PR2	35

10 Aggregate Functions

• It is possible to put arithmetic expressions in SELECT (select a,b a+b as sum from table). You can also use aggregate functions to compute summaries of data in a table. Most aggregate functions (except COUNT (*)) work on a single column of numerical data.

---- COUNT: The number of rows 不是计算总量是计算行数

---- SUM: The sum of the entries in the column

---- AVG: The average entry in a column

---- MIN, MAX: The minimum/maximum entries in a column

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
James	PR1	43
James	PR2	35
Jane	IAI	54

```
SELECT
   COUNT(*) AS Count
   FROM Grades;
```

Count
6

```
SELECT
   COUNT(Code) AS Count
   FROM Grades;
```

Count
6

```
SELECT
   COUNT(DISTINCT Code)
      AS Count
   FROM Grades;
```

Count
4

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
James	PR1	43
James	PR2	35
Jane	IAI	54

```
SELECT
   SUM(Mark) AS Total
   FROM Grades;
```

Total
320

```
SELECT
   MAX(Mark) AS Best
   FROM Grades;
```

Best
72

```
SELECT
   AVG(Mark) AS Mean
   FROM Grades;
```

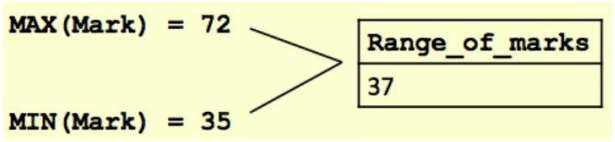
Mean
53.33

- You can combine aggregate functions using arithmetic

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
James	PR1	43
James	PR2	35
Jane	IAI	54

```
SELECT
 MAX(Mark) - MIN(Mark)
    AS Range_of_marks
 FROM Grades;
```



Hard example: Find John's average mark, weighted by the credits of each module

Modules

Code	Title	Credits
DBS	Database Systems	10
IAI	Introduction to AI	20
PRG	Programming	10

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60

Select sum(Grades.MARK*Modules.Credits)/sum(credits) as Final_mark from modules ,grades where Grades.Name ='John' and Modules.Code =Grades.Code;
*-- 不加就会输出一张名为sum(Grades.MARK*Modules.Credits)/sum(credits)的表*

```
Create DATABASE ex1;
use ex1;
create table modules(
        code varchar(255) PRIMARY key,
        title varchar(255),
        credits int
);
create table grades(
        name varchar(255),
        code varchar(255),
        mark int,
        CONSTRAINT g_m_fk Foreign key(code) references modules(code)
);

insert INTO modules VALUE('DBS','DATA BAESE',10),('1AI','1NTRO TO AI',20);
insert INTO GRADES VALUE('JOHN','DBS',56),('JOHN','1AI',72);
Select sum(Grades.MARK*Modules.Credits)/sum(credits) from modules ,grades
where Grades.Name ='John' and Modules.Code =Grades.Code;
```

• *You cannot use aggregate functions in the WHERE clause.But you can use them in the subquery in the WHERE clause.*

SELECT * FROM staff WHERE age > (SELECT AVG(age) FROM staff);

• *The use of aggregate functions leads to all rows after the first row being truncated.*

id	name	age
1	Annie	28
2	Brown	31
3	George	26
4	Kathy	21
5	Annie	21

```
SELECT *, AVG(age)
FROM staff;
```

id	name	age	AVG(age)
1	Annie	28	25.4000

11 GROUP BY

• *As the use of aggregate functions leads to all rows after the first row being truncated,but sometimes we want to apply aggregate functions to groups of rows. For example: find the average mark of each student individually. The **GROUP BY** clause achieves this.*

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
James	PR1	43
James	PR2	35
Jane	IAI	54

```
SELECT Name,
     AVG(Mark) AS Average
  FROM Grades
  GROUP BY Name;
```

Name	Average
John	64
Mary	60
James	39
Jane	54

• ***Group by*** 语句会根据指定的列将数据分为多个组，然后对每个组进行聚合操作，如求和、计数、平均值等操作，生成汇总结果。

```
SELECT Month, Department,
  SUM(Value) AS Total
FROM Sales
GROUP BY Month, Department;
```

Month	Department	Total
April	Fiction	60
April	Travel	25
March	Fiction	20
March	Technical	40
March	Travel	30
May	Fiction	20
May	Technical	50

```
SELECT Month, Department,
  SUM(Value) AS Total
FROM Sales
GROUP BY Department, Month;
```

Month	Department	Total
April	Fiction	60
March	Fiction	20
May	Fiction	20
March	Technical	40
May	Technical	50
April	Travel	25
March	Travel	30

• Group by 有先后顺序 前者为第一分类顺序，以此类推

• *Like aggregate functions, GROUP BY also truncates all rows after the first row for each sub-group.*

student_id	module	marks
1	CPT103	78
1	CPT111	81
2	CPT103	66
2	CPT111	63
2	CPT105	71

```
SELECT * FROM grades
GROUP BY student_id;
```

student_id	module	marks
1	CPT103	78
2	CPT103	66

12 Having

• HAVING 判断和 where 一致，但是用在 GROUP BY 中。它能查询 GROUP BY 分组后的结果进行条件过滤，允许我们筛选出符合指定条件的分组。

Grades

Name	Code	Mark
John	DBS	56
John	IAI	72
Mary	DBS	60
James	PR1	43
James	PR2	35
Jane	IAI	54

```
SELECT Name,
    AVG(Mark) AS Average
    FROM Grades
    GROUP BY Name
    HAVING
        AVG(Mark) >= 40;
```

Name	Average
John	64
Mary	60
Jane	54

Comparison:
*---- **Where** refers to the rows of tables, so cannot make use of aggregate functions.*
*---- **HAVING** refers to the groups of rows, and so cannot use columns or aggregate functions that does not exist after the step of column selection .*

SELECT [DISTINCT | ALL] columns FROM tables
[WHERE condition]
[ORDER BY column-list]
[GROUP BY column-list]
[HAVING condition]

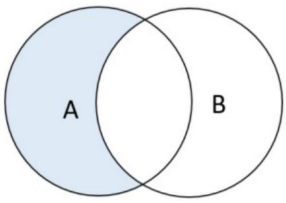
Operation process:
1. Tables are joined
2. WHERE clauses
3. GROUP BY clauses and aggregates
4. Column selection
5. HAVING clauses
6. ORDER BY

13 Index

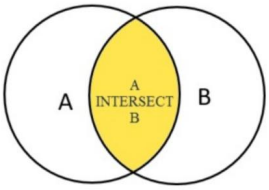
• *Why are Index needed: https://www.youtube.com/watch?v=fsG1XaZEa78*
---- An index helps to speed up select queries , but it slows down data input in update and insert statements.

• *Primary key and unique key affects the availability of certain functions of SQL.*
---- SQL 中，创建了主键或唯一键之后，会自动生成一个特殊的列名为 _rowid（或称 ROWID）。_rowid 列用于唯一标识表中的每一行数据,通过 _rowid 可以方便地定位和访问表中的数据行。没有定义主键或唯一键时，可能无法使用 _rowid 列提供的定位功能。

14 Set Operations -- UNION, INTERSECT and EXCEPT



A EXCEPT B



A INTERSECT B


A UNION B

• *We can treat the tables as sets and use the usual set operators of union, intersection and except to combine the results from two select statements.The results of the two selects should have the same columns and corresponding data types. (这个很重要不能想着生成俩个完全没有关系的列然后把他合并）Only UNION is supported in MySQL,and the other two can be simulated with subquery.*

Example: Find the average mark for each student and the average mark overall.in a single query.

SELECT Name, AVG(Mark) AS Average FROM Grades GROUP BY Name UNION SELECT 'Total' AS Name, AVG(Mark) AS Average FROM Grades;

Grades

Name	Code	Mark
Jane	IAI	54
John	DBS	56
John	IAI	72
James	PR1	43
James	PR2	35
Mary	DBS	60

Name	Average
James	39
Jane	54
John	64
Mary	60
Total	53.3333

15 How to deal with missing information ?

• Sometimes we don't know what value in a relation/table should have. Two main methods have been proposed to deal with this
*---- **NULL**s can be used as markers to show that information is missing .*
---- A default value can be used to represent the missing value.

15.1 NULL

*• **NULL** represents a state for an attribute that is currently unknown or is not applicable for this tuple. A method have been proposed to distinguish two types of **NULL**s:*
---- A-marks: data applicable but not known (for example, someone's age)
---- I-marks: data is inapplicable (telephone number for someone who does not have a telephone)

• We may face many problems when meeting nulls.Solution:When there are no NULLs around, conditions are evaluated into true or false. If a NULL is involved, a condition might be evaluated to 'undefined'/'unknown'.

a	b	a OR b	a AND b	a == b
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

---- WHERE clause of SQL SELECT uses three-valued logic(true/false/unknown): only tuples where the condition is evaluated to true are returned.

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

```
SELECT * FROM Employee
    Where Salary > 15,000;
```

Name	Salary
John	25,000
Anne	20,000

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

```
SELECT * FROM Employee
    Where Salary > 15,000
       OR Name = 'Chris';
```

Name	Salary
John	25,000
Anne	20,000
Chris	NULL

Citation: as the last one goes true in where!

---- Arithmetic operations applied NULLs changing into NULLS.

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Chris	NULL

```
SELECT Name,
 Salary * 0.05 AS Bonus
FROM Employee;
```

Name	Bonus
John	1,250
Mark	750
Anne	1,000
Chris	NULL

---- Aggregation of SQL return the value without computing NULL in sum(), avg() , min(),max(), count(),except count(*) which will return the actual table number of rows combing with all nulls

```
CREATE DATABASE EX1;
USE ex1;
CREATE TABLE MYTABLE(
      name varchar(255),
      salary int
);
INSERT into mytable VALUE
('john',25000),
('Mark',15000),
('Anne',20000),
('Chris',null);
INSERT into mytable VALUE(null,null);

SELECT * FROM mytable;
-- 'john',25000
-- 'Mark',15000
-- 'Anne',20000
-- 'Chris',null
-- null,null
```

SELECT count(name) AS Num FROM mytable;
-- 4
SELECT count(salary) AS Num FROM mytable;
-- 3
SELECT count(*) AS Num FROM mytable;
-- 5
SELECT sum(salary) AS sum FROM mytable;
-- 60000
SELECT sum(name) AS sum FROM mytable;
-- 0
SELECT AVG(salary) AS Avg FROM mytable;
-- 20000
SELECT AVG(name) AS Avg FROM mytable;
-- 0
SELECT min(salary) AS min FROM mytable;
-- 15000
INSERT into mytable VALUE(null,null);
SELECT count(*) AS Num FROM mytable;
-- 6

---- SQL NULLs are treated as equivalents in GROUP BY clauses

Employee

Name	Salary
John	25,000
Mark	15,000
Anne	20,000
Jack	NULL
Sam	20,000
Chris	NULL

```
SELECT Salary,
     COUNT(Name) AS Count
     FROM Employee
     GROUP BY Salary;
```

Salary	Count
NULL	2
15,000	1
20,000	2
25,000	1

---- SQL NULLs are considered and reported in ORDER BY clauses.

Continue from the upper code:
INSERT into mytable VALUE('Xu',-12);
SELECT * FROM mytable ORDER BY salary asc;

name	salary
Chris	
NULL	
NULL	
Xu	-12
Mark	15000
Anne	20000
john	25000

SELECT * FROM mytable ORDER BY salary desc;

name	salary
john	25000
Anne	20000
Mark	15000
Xu	-12
Chris	
NULL	
NULL	

• How to check for NULLs?
SELECT col FROM table WHERE col IS NULL;
SELECT col FROM table WHERE col IS NOT NULL;

15.2 Default Values: An alternative to the use of NULLs, which can have more meaning than NULLs. The followings are example default values of VARCHAR: 'none' / 'unknown' / 'not supplied'/ 'not applicable'. It is worth saying that not all defaults represent missing information and it depends on the situation.

Example:

Parts

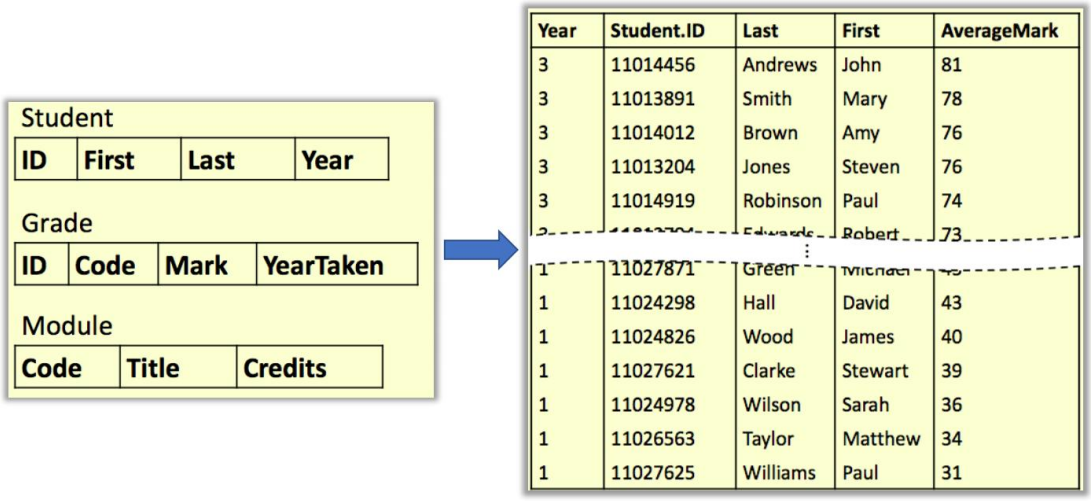
ID	Name	Weight	Quantity
1	Nut	10	20
2	Bolt	15	-1
3	Nail	3	100
4	Pin	-1	30
5	Unknown	20	20
6	Screw	-1	-1
7	Brace	150	0

---- Default values are 'Unknown' for Name and -1 for Weight and Quantity .
*---- There are still problems: if we use **UPDATE Parts SET Quantity = Quantity + 5***

16 Practice

• We want a list of students and their average mark: For first and second years the average is for that year ,and for finalists (third year) it is 40% of the second year plus 60% of the final year averages.

• We want the results: Sorted by year (high to low), then by average mark (high to low) then by last name, first name and finally ID. To take into account of the number of credits each module is worth. Produced by a single query



Year	Student.ID	Last	First	AverageMark
3	11014456	Andrews	John	81
3	11013891	Smith	Mary	78
3	11014012	Brown	Amy	76
3	11013204	Jones	Steven	76
3	11014919	Robinson	Paul	74
1	11027871	Green	[illegible]	[illegible]
1	11024298	Hall	David	43
1	11024826	Wood	James	40
1	11027621	Clarke	Stewart	39
1	11024978	Wilson	Sarah	36
1	11026563	Taylor	Matthew	34
1	11027625	Williams	Paul	31

思考：

Select * from ((Select student.year as year, student.id, last, first, sum(credits*mark)/sum(credits) as AverageMark from Student,Grade,Module where ~~(year=1 or year =2)~~ year in (1,2) and student.id = grade.id and Module.code=grade.code and year =yeartaken Group by year, student.id, last, first)

union

((Select student.year as year, student.id, last, first, sum(AverageMark) as AverageMark from ((Select student.year as year, student.id, last, first, 0.6*sum(credits*mark)/sum(credits) as AverageMark from Student,Grade,Module where year=3 and student.id = grade.id and Module.code=grade.code and yeartaken=3 Group by year, student.id, last, first)

union

(Select student.year as year, student.id, last, first, 0.4*sum(credits*mark)/sum(credits) as AverageMark from Student,Grade,Module where year=3 and student.id = grade.id and Module.code=grade.code and yeartaken =2 Group by year, student.id, last, first))) order by year desc,averagemark desc,last,first,Student.ID

问题：group by 一定要在每个应用 Aggregation of SQL 写起到分类作用

优化

1 把灰色合并：

Select student.year as year, student.id, last, first, sum(yeartaken*0.2*credits*mark)/sum(credits) as AverageMark from Student,Grade,Module where year=3 and student.id = grade.id and Module.code=grade.code and yeartaken in (2,3) Group by year, student.id, last, first

2 不需要在最外层写 select * from(.....),直接俩个 select union 连接后接 order by

----> go to lab2,3

3 Entity-Relationship Diagrams

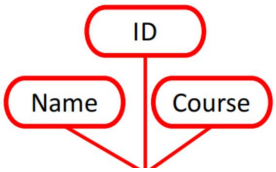
1 Introduction to ER model and diagram

• ***Database design*** *is important, which may result in a more efficient and simpler queries once the database has been created and may help reduce data redundancy in the tables.*

• ***ER Modelling*** *is used for conceptual design, which consists of three types of components:* ***Entities***, ***Attributes***, ***Relationships***. *ER Models are often represented as* ***ER diagrams*** *that gives a conceptual view of the database and identifies some problems in a design. There are various ways for representing ER diagrams,which specify the shape of the various components and the notation used to represent relationships.*

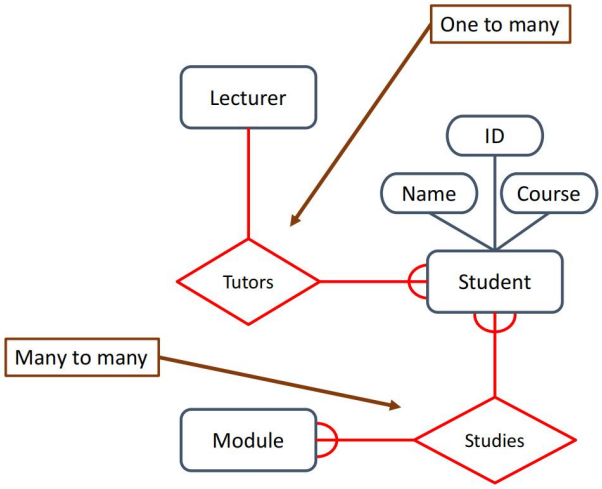
• ***Entities*** *represent objects or things of interest,and each entity is a general class, which has instances of that particular type(DBI and IAI are instances of Module) and various attributes (such as name, email address). In ER Diagrams, we will represent Entities as boxes with rounded corners. The box is labeled with the name of the class of objects represented by that entity.*

• ***Attributes*** *are properties about an entity. Each attribute have a name, an associated entity ,domains of possible values and the attribute values come from the domain. In ER Diagrams, attributes are drawn as ovals 椭圆形. Each attribute is linked to its entity by a line.*



• ***A relationship*** *is an association between two or more entities. Relationships have a name,a set of entities that participate in them, a degree(the number of entities that participate) and a cardinality ratio 基数比率. There are three types:*
----One to one (1:1) :Each lecturer has a unique office & offices are single occupancy
----One to many (1:M) /(1:):A lecturer may tutor many students, but each student has just one lecturer.*
----Many to many (M:M)/(M:N)/(:*):Each student takes several modules, and each module is taken by several students.*
Relationships are shown as links between two entities ,the name is given in a diamond box.(菱形) and the ends of the link(链接的俩端) show Cardinality among them. 注意下方爪子形状表示多，单线表示一在作画中不要省略

• *For example, in a University database we might have entities for Students, Modules and Lecturers. Students might have attributes such as their ID, Name, and Course. Students could have relationships with Modules (enrolment) and Lecturers (tutor/tutee).*



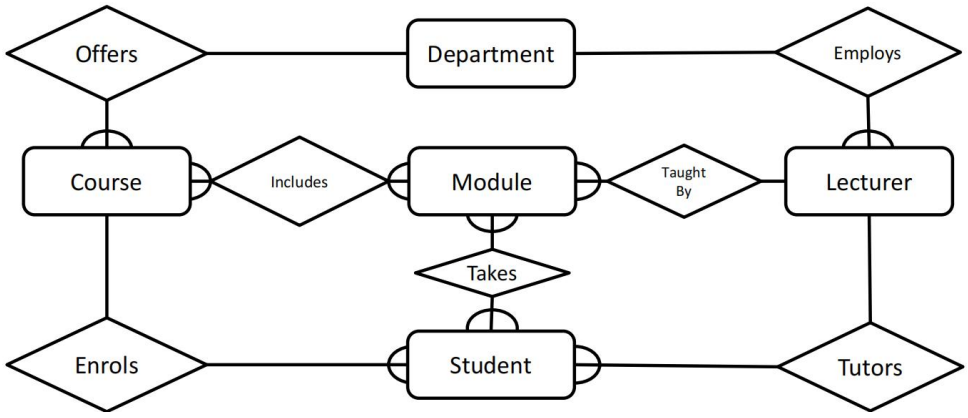
2 Design ER Models

2.1 Steps about drawing ER Models according to the description

• *Entities are things or objects they are often nouns in the description. Attributes are properties ,so they are often nouns.Relationships between entities are often described by verbs.*

E1: A university consists of a number of departments. Each department offers several courses. A number of modules make up each course. Students enroll in a particular course and take modules towards the completion of that course. Each module is taught by a lecturer from the appropriate department (several lecturers work in the same department), and each lecturer tutors a group of students. A lecturer can teach more than one module but can work only in one department.

Step1: find the all meaningful nouns and assume them as entities.
Step2: find the verb among this nouns.
Step3: draw the draft diagram（菱形中主语自行定义建议和动词一致）



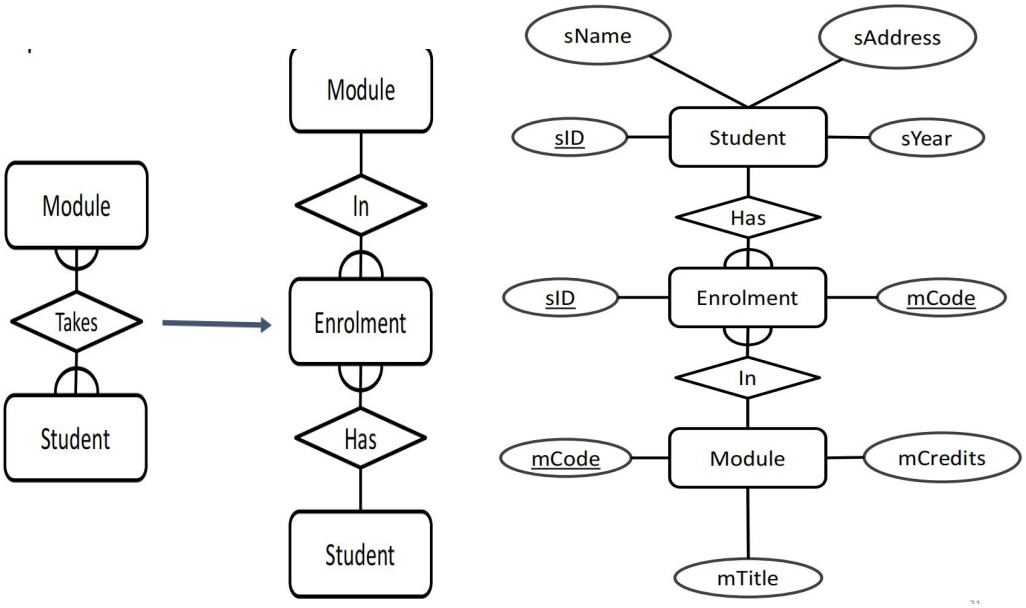
Step4: Remove the M:M relationship, as they are difficult to represent in a database.

Student (design 1)		
SID	**sName**	**sMod**
1001	Jack Smith	DBI
1001	Jack Smith	PRG
1001	Jack Smith	IAI
1002	Anne Jones	PRG
1002	Anne Jones	IAI
1002	Anne Jones	Vis

Module	
MID	**mName**
DBI	Databases and Interfaces
PRG	Programming
IAI	AI
VIS	Computer Vision

Student (design 2)		
SID	**sName**	**sMod**
1001	Jack Smith	DBI, PRG, IAI
1002	Anne Jones	VIS, IAI, PRG

----We can split a M:M relationship into two 1:M relationships. An additional entity is created to represent the M:M relationship.



----How to make the connections among those entities?

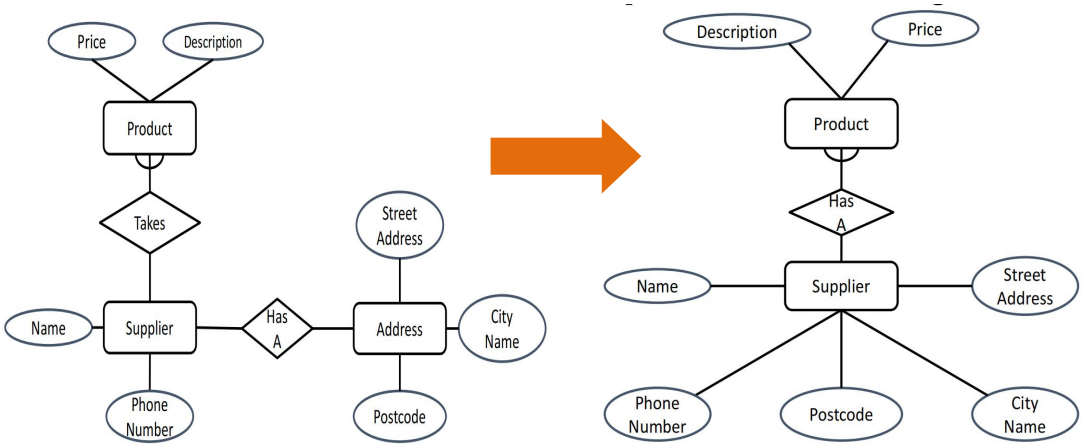
The Enrolment table will have columns for the student ID and module code attributes ,which will have a foreign key for student to represent the 'has' relationship and will have a foreign key for Module to represent the 'in' relationship.

----Comparison between Entities and Attributes

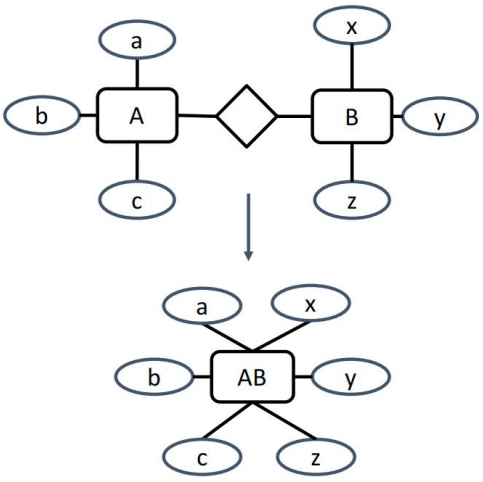
Sometimes it is hard to tell if something should be an entity or an attribute. Entities can have attributes but attributes have no smaller parts. Entities can have relationships between them, but an attribute belongs to a single entity.

Step5: Remove the 1:1 relationship.

E2: We want to represent information about products in a database. Each product has a description, a price and a supplier. Suppliers have addresses, phone numbers, and names. Each address is made up of a street address, a city name, and a postcode.



----Some relationships between entities might be redundant（冗余的）if it is a 1:1 relationship. We can merge the two entities that take part in a redundant relationship together. Then, they become a single entity and the new entity has all the attributes of the old ones.



2.2 From ER Diagram to SQL Tables

• Entities Become table names. Attributes of an entity becomes the columns. Relationships become foreign keys.(In ER Modelling, M:1 relationships between tables will become foreign keys.)

----> go to lab4

4 Normalisation

1 Introduction to Normalisation

• Characteristics of Good DB Designs

----The minimal number of attributes to support the data requirements of enterprise.

----Attributes have a close logical relationship.

----Minimal redundancy :each attribute represented only once ,except that those form all or part of foreign keys.

• Why minimal redundancy is important? Data redundancy not only increases memory usage, but also leads to update anomalies 异常.

Staff Branch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

---- Insert anomalies: One cannot add a new branch without the information of a staff in this branch.

---- Delete anomalies: Deleting the last staff of a branch also deletes the information of that branch.

---- Modification anomalies: To change one branch information, all rows in that branch must also be updated.

• Observe and find out the relationship between attributes to reorganize tables

----The staffNo refers to a unique staff member in the real life.As there's only one sName ,one staff position, one staff salary value, one branchNo and one bAddress for that staffNo. (Assume a staff can only have one position and works in only one branch office). There's only one staffNo associated with a sName? No There's only one staffNo associated with a position? No

==>staffNo has M:1 relationship with sName,position,salary, branchNo and bAddress.

----The branchNo refers to a unique branch office in the real life. There's only one unique bAddress associated with one branchNo. There's only one unique branchNo associated with one bAddress? Yes

==> branchNo has a 1:1 relationship with bAddress.

----The other attributes are not related in any aspects.

==>The StaffBranch table is better split into the Staff table and the Branch table.

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

• ***Normalisation*** *is a technique of reorganizing the data into multiple related tables in order to achieve minimal redundancy.*

• *Comparison between ER modeling and Normalisation:*
---- ER diagram is useful when you have detailed database specifications, but no existing table design.
---- When you already have a database, but the tables are not well designed, you can use normalisation techniques to improve them.

2 Functional Dependency

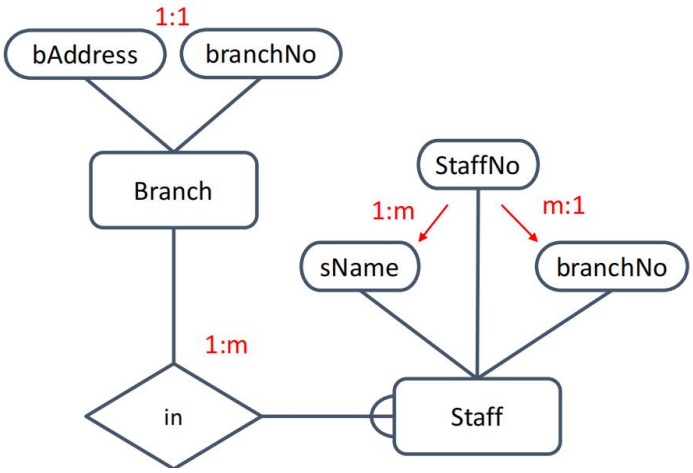
• ***Functional dependency (FD)****: If A and B are attribute sets of relation R,* ***B is functionally dependent on A (denoted A → B)****, if each value of A in R is associated with exactly one value of B in R. In this way, A is called* ***determinant****. The determinant has a M:1 or 1:1 relationship with other attributes in Functional dependency.*

Example: PersonalTutors{LecID, LName, LEmail, StudentID, SName}
LecID → LName, Lemail
StudentID → Sname
LName → LecID
StudentID → LecID
LEmail → LecID, LName
----If these attributes are put together, they can form a relation and the determinant becomes the unique key or primary key of the relation.(LecID, LName, LEmail) is a super key,which is not a good choice.
StudentID → SName, LEmail
LEmail → LecID, LName

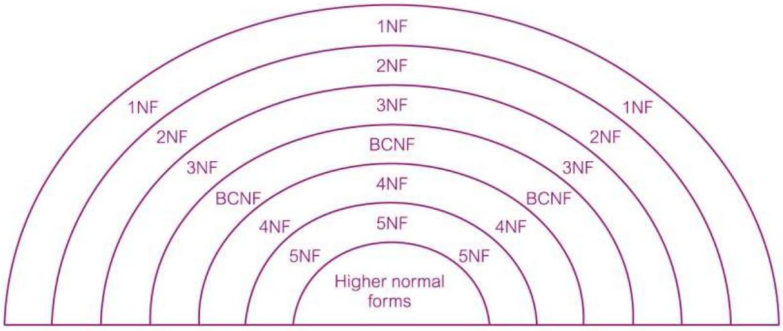
• ***Full functional dependency****: If A and B are two sets of attributes of a relation, B is fully functionally dependent on A, if B is functionally dependent just on A (A → B), not on any proper subset of A. In other words, determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the right hand-side.*
Partial functional dependency*:For A → B, if C is a proper subset of A (C ⊂ A) and C → B,then B is partially functionally dependent on A.*

---- We only care about full functional dependency in normalisation. Because partial functional dependency results in super keys. If you use a foreign key to refer to a super key in another table, you will need extra columns in the referencing table.

• For attributes that have a M:1 relationships, if they belong to the same context, they will be grouped into one relation. Otherwise, they can be split into two tables and be linked with a foreign key.



3 Normal Forms



*• A relation is in **unnormalised Form (UNF)** if some data values are not **atomic**, which means column values should be single values, not composite objects.*

• Problems with UNF :To update/delete the textbook name 'T1' , you need to manually update do so.

Unnormalised relation

Module	Dept	Lecturer	Texts
M1	D1	L1	T1, T2
M2	D1	L1	T1, T3
M3	D1	L2	T4
M4	D2	L3	T1, T5
M5	D2	L4	T6

• *A relation is said to be in **first normal form (1NF)** if all data values are **atomic.***

• *Problems with 1NF:*

---- Changing module code from 'M1'to something else requires you to check the whole table.

---- Adding a new lecturer with no modules and text is impossible.

---- If a (department, lecturer) pair is modified, the change must be made to all (Module, Text) pairs. For example, to change (D1, L1) to some other value, you must manually change the first four rows.

---- If (M3, T4) is deleted, L2 will be permanently lost.

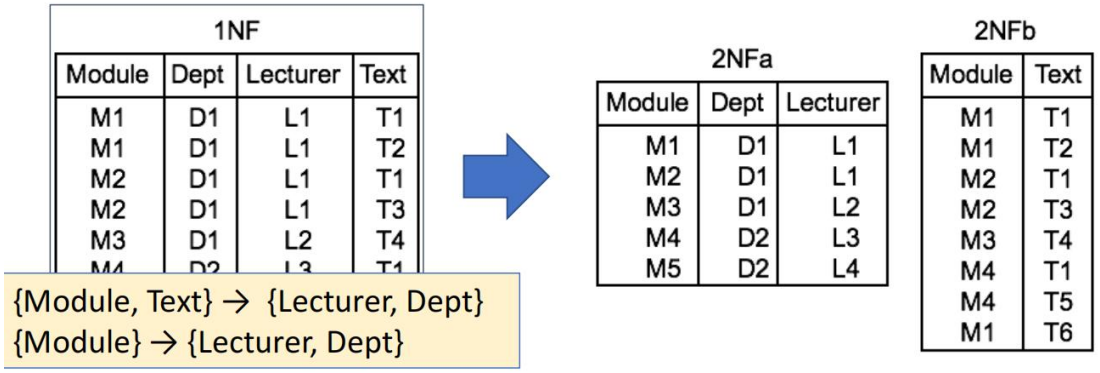
1NF

Module	Dept	Lecturer	Text
M1	D1	L1	T1
M1	D1	L1	T2
M2	D1	L1	T1
M2	D1	L1	T3
M3	D1	L2	T4
M4	D2	L3	T1
M4	D2	L3	T5
M5	D2	L4	T6

• *A relation is in **second normal form (2NF)** if it is in 1NF and no non-key(非主键) attribute is partially dependent on the primary key.*

• *For example,the upper picture has {Module, Text} →{Lecturer, Dept}. But also has {Module} → {Lecturer, Dept}. Therefore, Lecturer and Dept are partially dependent on the primary key and the table is in 1NF , not in 2NF.*

• *1NF to 2NF*



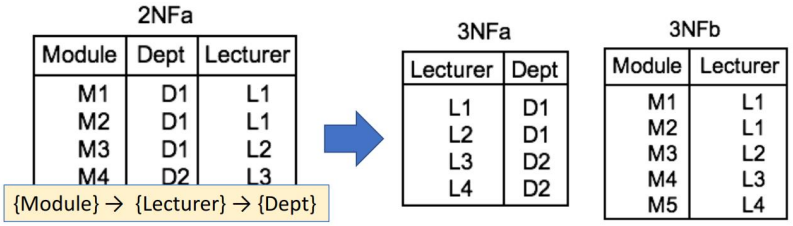
• *Problems in 2NF*

---- When deleting M3, we lose L2 forever.

---- To change the department for L1, we need to change multiple rows manually.

• ***Third Normal Form (3NF)*** *: A relation that is in 1NF and 2NF and in which no non-key attribute is* ***transitively dependent*** *on the primary key.*

- *2NF to 3NF*



4 Transitive Dependency

• ***Transitive dependency*** *describes a condition where A, B and C are attributes of a relation such that if A→B and B→C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C,which means B→A or C→A is not true).*

Example:
staffNo (A) → branchNo (B) → bAddress (C)==> This is transitive dependency.
LecID (A) → LEmail (B) → LName (C)==>This is not a transitive dependency.
Because LEmail (B) → LecID (A), which breaks the definition of transitive Dependency.

• Why the second example should not be split into two tables? Because after splitting them,we will get Table1: LecID, LEmail and Table2: LEmail, LName. Table1 and Table2 has 1:1 relationship, with LEmail being the foreign key. According to ER modelling, this is redundant.

5 Practice

{order, product, customer, address, quantity, unitPrice} For example: (001, Laptop, Jianjun, SEB435 UNNC, 1, $500). The primary key is {order, product}. 这里一个订单可以有多个产品.Please normalize it to 3NF.
{product, unitPrice}
{order 订单, product, quantity}
{order, customer}
{customer, address}

6 SQL Support for Normalisation

INSERT INTO branch SELECT DISTINCT branchno, baddress FROM staffbranch;
INSERT INTO staff SELECT staffno, …, branchno FROM staffbranch;
CREATE TABLE branch SELECT DISTINCT branchno, baddress FROM staffbranch
CREATE TABLE staff SELECT staffno, …, branchno FROM staffbranch.

5 Transactions and Recovery

• ***A transaction*** *is an action or a series of actions, carried out by a single user or an application program, which reads or updates the contents of a database. It is logical units of work on a database,which achieve recovery, consistency and integrity. Each unit does something in the database and it achieves nothing of use or interest separately.*

More information about SQL can be found at :
https://dev.mysql.com/doc/refman/8.0/en/commit.html

```
START TRANSACTION
    [transaction_characteristic [, transaction_characteristic] ...]

transaction_characteristic: {
    WITH CONSISTENT SNAPSHOT
  | READ WRITE
  | READ ONLY
}

BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

• *Transaction have ACID properties:* ***Atomicity , Consistency, Isolation, Durability.***
---- Atomicity: a transaction can't be executed partially.
---- Consistency 一致性: Transactions take the database from one consistent state into another and it does not exist the other condition.
---- Isolation: The effects of a transaction are not visible to other transactions until it has completed and the transaction has either happened or not from outside.
---- Durability: Once a transaction has completed, its changes are made permanent. 永久的 Even if the system crashes, the effects of a transaction must remain in place.

• *The* ***transaction manager*** *enforces the ACID properties. It schedules the operations of transactions :****COMMIT*** *and* ***ROLLBACK*** *are used to ensure atomicity.* ***Locks*** *are used*

to ensure consistency and isolation for concurrent transactions. ***A log*** 日志 *is kept to ensure durability in the event of system failure.*