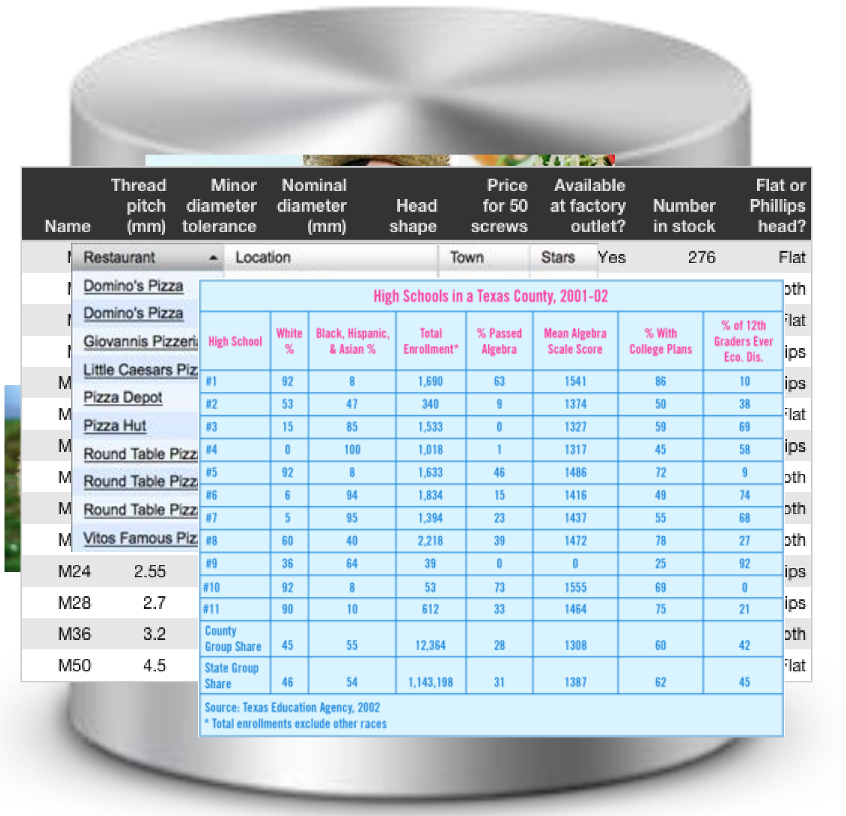
Principles of Database Systems

Review

主讲教师：
薛昕惟
办公室：信息楼320B
xuexinwei@dlut.edu.cn

Basic Concepts

- Database (DB)
  - Collection of **interrelated** data



High Schools in a Texas County, 2001-02

High School	White %	Black, Hispanic, & Asian %	Total Enrollment*	% Passed Algebra	Mean Algebra Scale Score	% With College Plans	% of 12th Graders Ever Eco. Dis.
#1	92	8	1,690	63	1541	86	10
#2	53	47	340	9	1374	50	38
#3	15	85	1,533	0	1327	59	69
#4	0	100	1,018	1	1317	45	58
#5	92	8	1,633	46	1486	72	9
#6	6	94	1,834	15	1416	49	74
#7	5	95	1,394	23	1437	55	68
#8	60	40	2,218	39	1472	78	27
#9	36	64	39	0	0	25	92
#10	92	8	53	73	1555	69	0
#11	90	10	612	33	1464	75	21
County Group Share	45	55	12,364	28	1308	60	42
State Group Share	46	54	1,143,198	31	1387	62	45

Source: Texas Education Agency, 2002
* Total enrollments exclude other races

Additional issues

- Some other definitions
  - Database(DB):a collection of interrelated data , stored in systems as files (データベース)
  - Database management system (DBMS): a system/mechanism to manage data in DB
    or: set of programs to access the data in DB
    (データベース管理システム)
  - Database system(DBS): DB + DBMS + Users/Administers (データベースシステム)
  - Database application system: DB + DBMS + Application programs + Users/Administers
    (データベースアプリケーションシステム)

Drawbacks of using file systems to store data

- Data redundancy and inconsistency (数据冗余和不一致)
- Difficulty in accessing data(数据访问困难)
- Data isolation (数据孤立)
- Integrity problems (完整性问题)
- Atomicity of updates (更新操作的原子性)
- Concurrent access by multiple users (多用户并发访问)
- Security problems (安全性问题)

Database systems offer solutions to all the above problems

事务的特性

- 事务的ACID特性：
  - 原子性（**Atomicity**）

    两个操作要么全做，要么全不做

A	B
A=A-1	
	B=B+1

  - 一致性（**Consistency**）

    全做或者全不做，数据库都处于一致性状态
  - 隔离性（**Isolation**）

    对并发执行而言，一个事务的执行不能被其他事务干扰
  - 持续性（**Durability** ）

    一个事务一旦提交，它对数据库中数据的改变就应该是永久性的。

并发执行

- 事务并发执行带来的问题
  - 可能会存取和存储不正确的数据，破坏事务的隔离性和数据库的一致性
  - **DBMS必须提供并发控制机制**
  - 并发控制机制是衡量一个DBMS性能的重要标志之一

T_1	T_2
① 读A=16	
②	读A=16
③ A←A-1 写回 A=15	
④	A←A-3 写回A=13

T1的修改被T2覆盖了！

封锁协议

T_1	T_2
① Xlock A	
获得	
② 读A=16	
	Xlock A
③A←A-1	等待
写回A=15	等待
Commit	等待
Unlock A	等待
④	获得Xlock A
	读A=15
	A←A-1
⑤	写回A=14
	Commit
	Unlock A

没有丢失修改

Drawbacks of using file systems to store data

- Data redundancy and inconsistency (数据冗余和不一致)
- Difficulty in accessing data(数据访问困难)
- Data isolation (数据孤立)
- Integrity problems (完整性问题)
- Atomicity of updates (更新操作的原子性)
- Concurrent access by multiple users (多用户并发访问)
- Security problems (安全性问题)

Database systems offer solutions to all the above problems

View of Data

Data Abstraction

- For the system to be usable, it must retrieve data efficiently. (高效的检索数据)

- The need for efficiency has led designers to use **complex data structures** （数据结构） to represent data in the database.

- Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system.

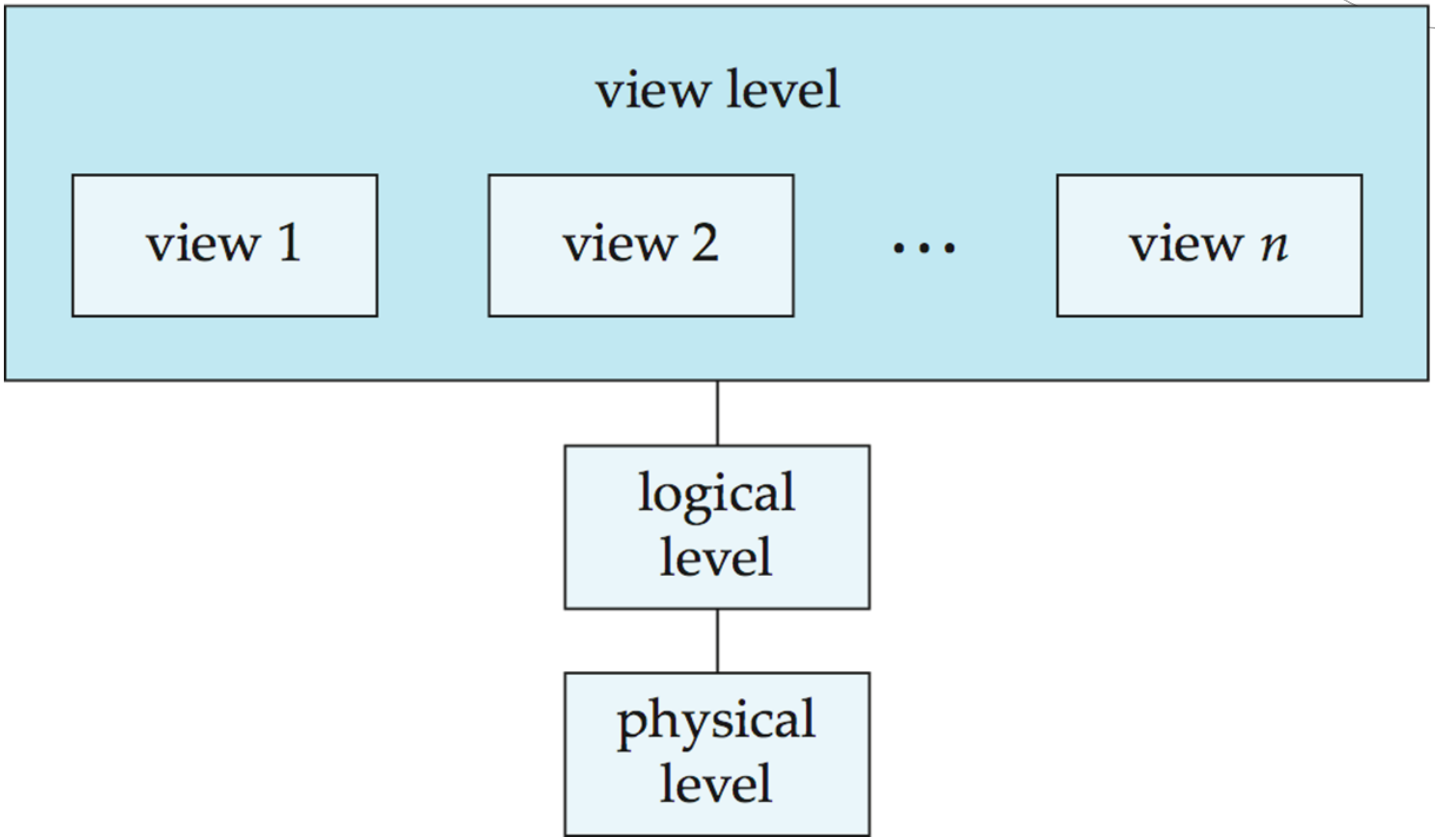
View of Data

- A major purpose of a database system is to provide users with an **abstract view of the data**(抽象的数据视图).

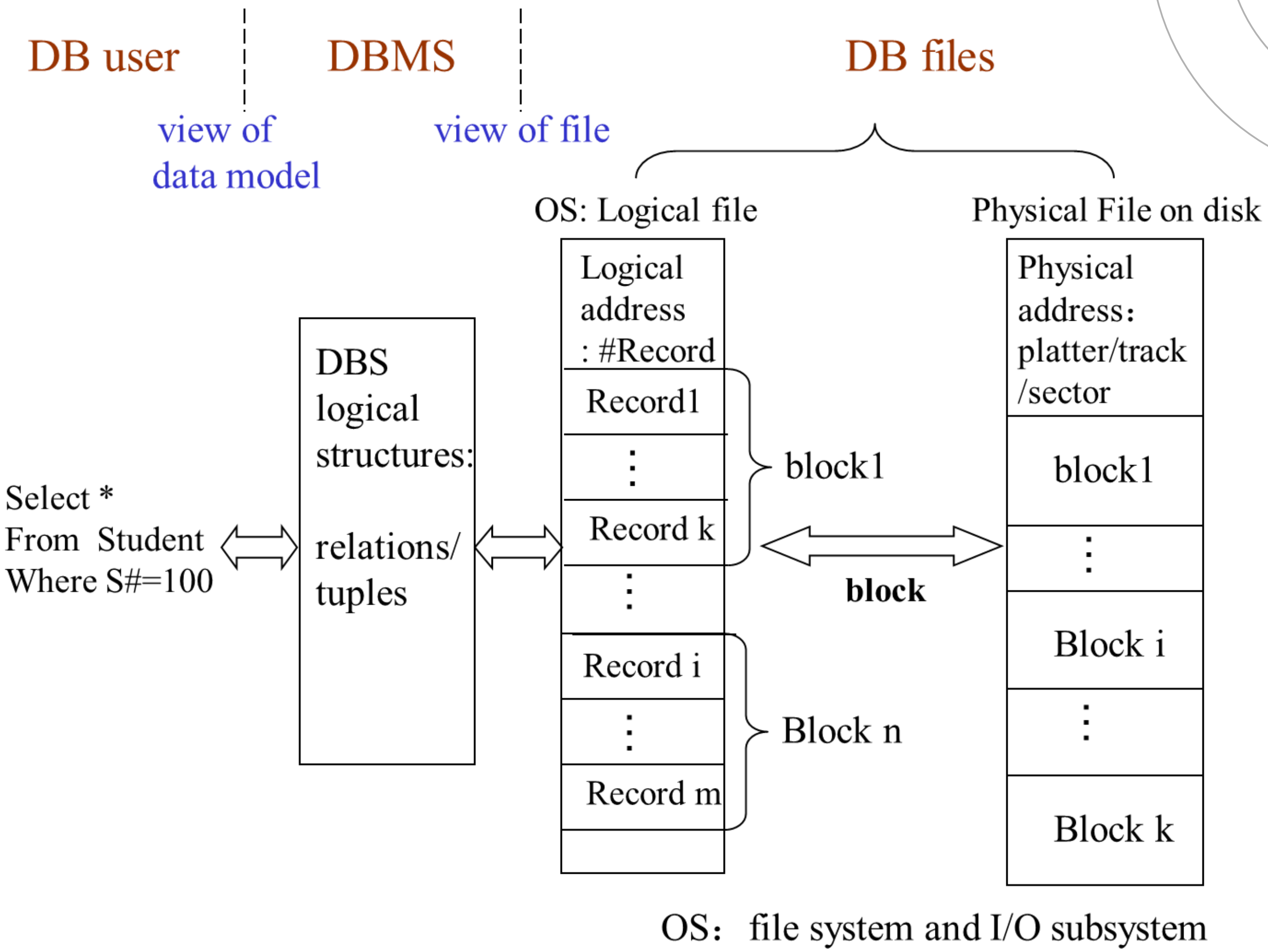
- That is, the system **hides certain details** of how the data are stored and maintained.(隐藏存储细节)

Data Abstraction

- An architecture for a database system



Data Abstraction



Keyword

- View of Data
  - Data Abstraction
    - architecture for a database system
      - Physical level
      - Logical level
      - View level
  - Schema (スキーマ)
    - Physical schema (内部スキーマ)
    - Logical schema　(概念スキーマ）
    - Subschema　(外部スキーマ）
  - Physical Data Independence (物理的データ独立性)

Data Model (数据模型)

- Data descriptions/abstractions in three levels must obey three types of specification, i.e. three types of **data models**

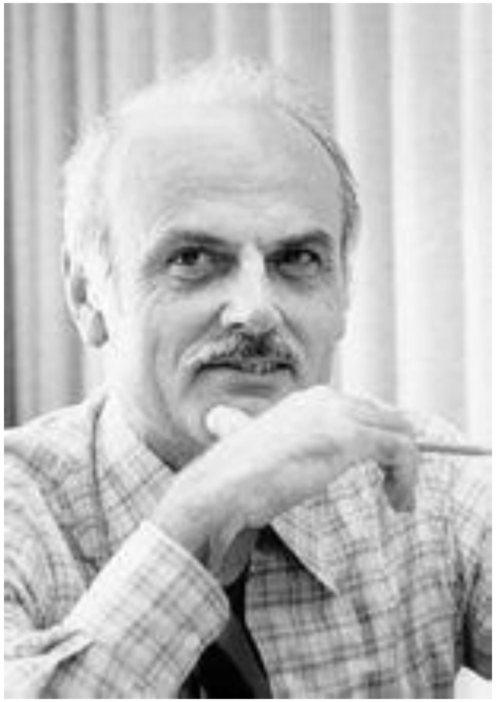
- Definition of data model:
  - a collection of conceptual tools for describing
    - data
    - data relationships (数据联系)
    - data semantics (数据语义)
    - consistency constraints (一致性约束)

- A data model provides a way to describe the design of a database at the physical, logical, and view levels.

Data Model (数据模型)

- In the course, data models can be classified as
  - *Conceptual Data Model（概念数据模型）*
    - *Entity-Relationship Model (实体-联系模型)*
  - *Logical Data Model（逻辑数据模型）*
    - ***Relational model (关系模型)***
    - *Network data model (网状模型)*
    - *Hierarchical data model (层次模型)*
    - *Object-based data model (基于对象的数据模型)*
    - *Semistructured data model (半结构化数据模型)*
  - *Physical Data Model（物理数据模型）*
    - *B* tree model…*

Relational Model



埃德加·弗兰克·科德（Edgar Frank Codd，1923－2003），“关系数据库之父”，1981年获图灵奖。

1970年，科德发表题为“**A Relational Model of Data for Large Shared Data Banks(大型共享数据库的关系模型)**”的论文，文中首次提出了数据库的关系模型。

Relational Model

- Relational data structure
- Integrity constraints
  - constraints on attributes of schemas, e.g. value domain, type （域完整性）
  - constraints on dependencies among attributes of a schema （实体完整性）
  - constraints on dependencies among attributes of different schemas （参照完整性）
- Operations on the model

关系/元组/属性/域/笛卡尔积

元组/记录/行

姓名	生日	身高	项目	时间	国家号
博尔特	1986.8.21	196	100米跑	9’58	1
苏炳添	1989.8.29	172	100米跑	9’83	2
张雨霏	1998.4.19	176	100米蝶	55’64	2

属性/字段/列

编号	国家名
1	牙买加
2	中国
3	美国

Terms-Formal Definitions

- Note:
  - Relation *r* in Database field is the limited set (有限集合)
  - Attributes in tuples are non-ordered (无序性)
    - e.g. $(d_1, d_2, \ldots, d_n) = (d_2, d_1, \ldots, d_n)$
  - the order in which the tuples appear in a relations is irrelevant
  - several attributes may have the same domain

Terms-Formal Definitions

- Note:
  - A domain is **atomic** (原子的) if its elements are considered to be indivisible
  - all domains of R **should be atomic**, first formal norm
    - e.g. multivalued attribute values are not atomic, { {1,2}, {4}, {4,6,7} }
    - e.g. composite attribute address={city, street, zipcode} is not atomic(or atomic?)
  - **The important issue is not what the domain itself is, but rather how we use domain elements in our database.**

Keys

- Superkey （スーパーキー）
- Candidate key （候補キー）
- Primary key （主キー）
- Foreign key （外部キー）
- Referential integrity constraint
- Entity integrity constraint

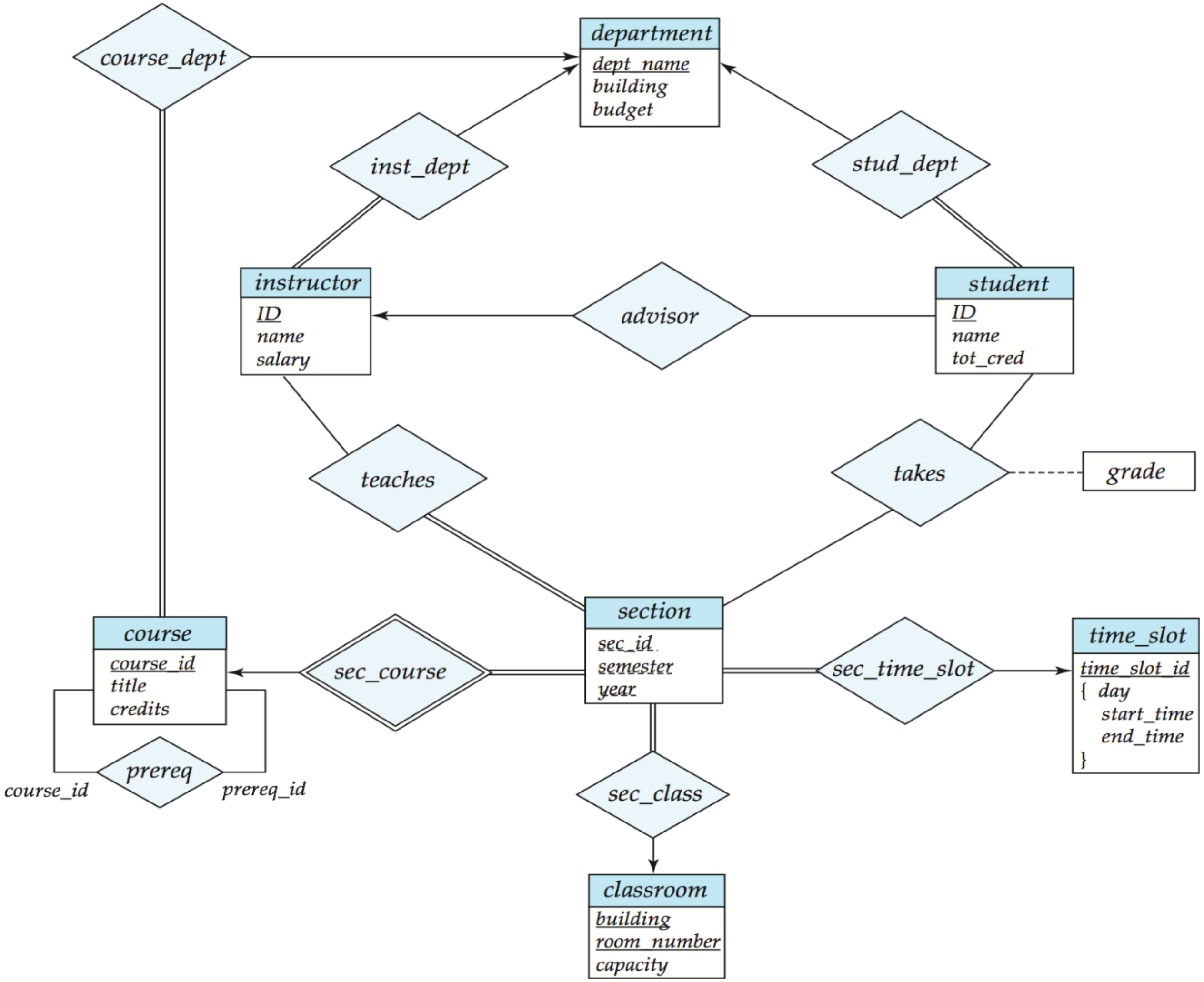
Overview of the Design Process

Design Phases

- characterize the data needs
- conceptual-design
  - entity-relationship model
  - specification of functional requirements(功能需求规格说明)
- implementation of the database
  - logical-design phase
  - physical-design phase

Database Design and the E-R Model

E-R diagram for the University Enterprise



Features of Good Relational Designs

Think…

- Which is better?

instructor(*ID*, *name*, *dept name*, *salary*)
department(*dept name*, *building*, *budget*)

VS

inst dept (*ID*, *name*, *salary*, *dept name*, *building*, *budget*)

Normal Forms

- A relation schema *R* is **in first normal form** (1NF) if the domains of all attributes of *R* are atomic.

学号	姓名	院系	系主任	课程号	得分
sno	**sname**	**dept**	**dleader**	**cno**	**score**
S1	张翠山	武当派	张三丰	C1	90
S1	张翠山	武当派	张三丰	C2	80
S2	谢逊	明教	张无忌	C3	100
S2	谢逊	明教	张无忌	C3	90

Boyce–Codd Normal Form

- **Boyce–Codd normal form** (**BCNF**), eliminates all redundancy that can be discovered based on functional dependencies
  - BCNF消除所有基于函数依赖能够发现的冗余
- A relation schema R is in BCNF with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
  - $\alpha \rightarrow \beta$ is a trivial functional dependency (that is, $\beta \subseteq \alpha$).
  - α is a superkey for schema R.

Boyce–Codd Normal Form

- We now state a general rule for decomposing(分解) that are not in BCNF.
  - Let R be a schema that is not in BCNF.
  - Then there is at least one nontrivial functional dependency $\alpha \rightarrow \beta$ such that α is not a superkey for R.
  - We replace R in our design with two schemas:
    - $(\alpha \cup \beta)$
    - $(R - (\beta - \alpha))$

Third Normal Form

- A relation schema R is in **third normal form** with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, at least one of the following holds:
  - $\alpha \rightarrow \beta$ is a trivial functional dependency.
  - α is a superkey for R.
  - **Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.**

Database Languages

Database Languages

- Database Languages as human-machine interfaces
  - **Data-Manipulation Language, DML** (数据操纵语言)
  - **Data-Definition Language, DDL** (数据定义语言)

姓名	生日	身高	项目	时间	国家
博尔特	1986.821	196	100米跑	9′79	牙买加
苏炳添	1989.8.29	172	100米跑	~~9′99~~ 9’83	中国
张雨霏	1998.4.19	176	100米蝶	55′64	中国
菲尔普斯	1985.6.30	193	100米蝶	50′58	美国

Database Languages

- Data Definition Language (DDL)
  - DDL is used for specifying the database schema and additional properties of the data
    - E.g.

```
create table account (
        account-number    char(10),
        balance           integer);
```

  - DDL can also be used to define integrity constraints in DB
    - domain integrity（域约束）, referential integrity（参照完整性）, assertions（断言）, authorization（授权）, etc.

Database Languages

- Data Definition Language (DDL)
  - just like any other programming language, the DDL gets as input some instructions (statements) and generates some output.
  - The output of the DDL is placed in the **data dictionary(数据字典)**, which contains **metadata(元数据)**

Database Languages

- Metadata: data about data
  - The structures /**schemas** of the database defined by DDL
  - **Integrity constraints** (完整性约束)
  - **Primary key** (主键) (ID uniquely identifies instructors)
  - **Referential integrity** (参照完整性) (references constraint in SQL)
    - e.g. dept_name value in any instructor tuple must appear in department relation
  - **Authorization** (授权)

Database Languages

- **Data Manipulation Language (DML)**
  - Language for **accessing** and **manipulating** the data organized by the appropriate data model
  - DML also known as query language

- Two classes of languages
  - **Procedural (过程化DML)**– user specifies what data is required and how to get those data
  - **Declarative (nonprocedural) (声明式DML)** – user specifies what data is required without specifying how to get those data

- Query (查询):a statement requesting the retrieval of information

- SQL is the most widely used query language

Database Languages

- SQL: widely used non-procedural language
  - Example: Find the name of the instructor with ID 22222

```
select  name
from    instructor
where   instructor.ID = '22222'
```

  - Example: Find the ID and building of instructors in the Physics dept.

```
select instructor.ID, department.building
from instructor, department
where instructor.dept_name=department.dept_name
  and department.dept_name = 'Physics'
```

SQL Data Definition

Basic Schema Definition-Create

- The general form of the **create table** command is:

  create table r
  $(A_1\ D_1,$
  $A_2\ D_2,$
  $\ldots,$
  $A_n\ D_n,$
  $(\text{integrity-constraint}_1), \ldots,$
  $(\text{integrity-constraint}_k));$

  – r is the name of the relation
  – each A_i is an attribute name in the schema of relation r
  – D_i is the data type of values in the domain of attribute A_i

Integrity Constraints on a Single Relation

- primary key
- not null
- unique
- foreign keys
- check (P), where P is a predicate

Cascading Actions in Referential Integrity

- When the DB is modified by **Insert, Delete,** and **Update**, the tests must be made in order to preserve the referential integrity constraint.

- **create table** course (

  …

  dept_name **varchar**(20),

  foreign key (dept_name) **references** department

  on delete cascade

  on update cascade,

  . . .

)
- alternative actions to **cascade**: **set null**, **set default**

Assertions

- E.g. The value of the attribute tot_cred for each student must equal the sum of credits of courses that the student has completed successfully.

create assertion credits_earned_constraint **check**
(not exists (select ID **from** student
where tot_cred < > **(**
select sum(credits)
from takes join course
on takes.course_id= course.course_id
where student.ID=takes.ID **and** grade **is not null and** grade < > 'F'**)**

Basic Schema Definition-Drop

- The **drop table** command **deletes all information (tuples and schema)** about the dropped relation from the database

drop table *r*

Basic Schema Definition-Alter

- The **alter table** command is used to add or delete/drop attributes to an existing relation

alter table *r* **add** *A D*

where *A* is the name of the attribute to be added to relation *r* and *D* is the type of *A*

- all tuples in the relation are assigned null as the value for the new attribute

alter table *r* **drop** *A*

where *A* is the name of an attribute of relation *r*

- dropping of attributes not supported by many databases

Views

- SQL provides the **create view** command to create a view, which is stored in the database **in the long run**
  - **create view** v **as** <query expression>
  - where
    - <query expression> is any legal expression
    - the **view** name is represented by v

- View definition is not the same as creating a new relation by evaluating the query expression. (创建视图与创建关系不同)
  - Rather, a view definition causes the **saving of an expression**; the expression is substituted into queries using the view. (存储的是表达式)

SQL Data Manipulation

Data Manipulation

- A newly created table is empty initially, we can use **insert** command to load data into the table

  insert into *instructor*
  values (10211, 'Smith', 'Biology', 66000);

- The **delete** command removes tuples from the table

  delete from account

- The **update** command changes a value in a tuple without changing all values in the tuple.

  update instructor
  set salary= salary * 1.05;

SQL Data Query

SQL query

- A typical SQL query has the form:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P
  group by $A_1, A_2 \ldots\ldots$
  having P
  order by A_1, A_2

①	②	③	④	⑤	⑥
FROM	WHERE	GROUP BY	HAVING	SELECT	ORDER BY

SELECT

- select * from 社員;
- select 氏名,性別 from 社員;
- select 氏名 from 社員 where 性別 = '男';
- select 氏名, 年齢 from 社員where 年齢 >= 45;
- select * from 社員 where 氏名 like ‘田%’;
- select 氏名 from 社員 where (年齢 < 40) and (性別 = '女');
- select 氏名 from 社員 where 性別 = ’男’ order by 氏名よみ;
- select distinct 年齢 from 社員 where 性別 = '男';

SELECT

- select count(*), avg(基本給), sum(基本給),max(基本給), min(基本給) from 社員;
- select 性別, avg (基本給) from 社員 group by 性別;
- select 性別, avg(基本給) from 社員 where 年齢 <= 40 group by 性別 having avg(基本給) > 100000;

Aggregate Functions

- Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value.
  - Average: **avg**
  - Minimum: **min**
  - Maximum: **max**
  - Total : **sum**
  - Count: **count**

Aggregation with Null Values

- All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes (除了count(*)外所有的聚集函数都忽略输入集合中的空值)
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

Joined Relations

- **Join operations** take two relations and return as a result another relation.
- These additional operations are typically used as subquery expressions in the **from** clause
- **Join condition** (连接条件)– defines which tuples in the two relations match, and what attributes are present in the result of the join.
- **Join type**(连接类型) – defines how tuples in each relation that do not match any tuple in the other relation (based on the join condition) are treated.

Join types
inner join
left outer join
right outer join
full outer join

Join Conditions
natural
on <predicate>
using $(A_1, A_1, \ldots, A_n)$

Set Operations(集合运算)

Set Operations

- Set operations **union**(并), **intersect** (交), and **except** (差)

- Each of the above operations automatically eliminates duplicates(重复)

- To retain all duplicates use the corresponding multiset(多重集) versions **union all**, **intersect all** and **except all**.
  - MS SQL Server doesn't support **intersect all** and **except all**

Nested Subqueries
(嵌套子查询)

Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries.
- A **subquery** is a select-from-where expression that is nested within another query. (子查询是嵌套在另一个查询中的select-from-where表达式)

```
select ・・・ from ・・・
where ・・・・
      属性名 =
   (select 属性名 from ・・・
                where ・・・);
```

主查询　　　　子查询

Nested Subqueries 1

```
SELECT couid, score
FROM selection
WHERE stuid =
    (SELECT stuid
     FROM student
    WHERE stuname = 'Inoue');
```

Nested Subqueries 2

SELECT stuname FROM student WHERE stuid IN (SELECT stuid FROM selection WHERE score = 'A');

- Aの成績のある科目を受講している学生の学生名を知りたい副問合せの結果の中が複数行である場合にはin述語を用いる。

Nested Subqueries 3

```
SELECT * FROM student WHERE age > =ALL (
        SELECT age FROM student WHERE stuid IN (
         SELECT stuid FROM selection WHERE score = ‘A’)
);
```

Nested Subqueries 4

```
SELECT * FROM student WHERE EXISTS (
        SELECT * FROM selection WHERE score = 'A'
        AND stuid = student.stuid
);
```

Nested Subqueries 5

```
select distinct S.ID, S.name
from student as S
where not exists ( (select course_id
                    from course
                    where dept_name = 'Biology')
                   except
                   (select T.course_id
                     from takes as T
                     where S.ID = T.ID) );
```

Nested Subqueries 6

```
select dept_name, avg_salary
from (select dept_name, avg (salary)
        from instructor
        group by dept_name)
         as dept_avg (dept_name,  avg_salary)
where avg_salary > 42000;
```

Nested Subqueries 7

```
with dept_total (dept_name, value) as
      (select dept_name, sum(salary)
       from instructor
       group by dept_name),
     dept_total_avg(value) as
     (select avg(value)
      from dept_total)
select dept_name
from dept_total, dept_total_avg
where dept_total.value >= dept_total_avg.value;
```

Relational Algebra

- The relational algebra is **procedural** query language
  - a set of operations, take one or two limited relations as input and produce a new limited relation as output

- Three types of operations/operators on relations
  - **fundamental operations**（基本运算）
  - **additive operations**（附加运算）
  - **extended operations**（扩展运算）

Fundamental Operations

- Six basic operators
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - Cartesian product: x
  - rename: ρ

Unary operations (select, project)

Binary operations (union, set difference, Cartesian product)

Unary operations (rename)

Additional Operations

- We define additional operations that do not add any power to the relational algebra, but that **simplify common queries**.
  - **Set Intersection**
  - **Natural Join**
  - **Division**(除)
  - **Assignment**(赋值)
  - **Outer Join**

- Note: An additional operation can be replaced/re-represented by basic operations.

Extended Relational-Algebra-Operations

- We define extended operations that **add power** to the relational algebra
  - **Generalized Projection**(广义投影)
  - **Aggregation**(聚集)

- Note: An extended operation cannot be expressed using the basic relational-algebra operations.

Thanks