Isys2120 Data and Information Management

Week 1

**Outline**

• Data and DBMS in context

• The relational data model

– Terminology

– Practice understanding relational data

• Careers and people’s roles

• DBMS versus other ways to manage shared data

• Some big ideas

Terminology

• Data: Facts that can be recorded

• Database: A collection of data, is persistant and shared

• Database Management System (DBMS): software package designed to store and manage one or more databases

• Meta-data: descriptions of the format of the data

Relational DBMS: data is stored as instance, containing different entries, and matching data indicates connection.

Instance: contents of the database at a single time

Schema: the structure of data in a particular database( what tables, what metadata(column name))

**Data practice**

**Data design**: to decide what schema will be used for the data.

It proceed in stages:

– first produce a conceptual or semantic model,

– then translate this into a relational schema,

– then evaluate the schema for quality, and improve it if needed

**Languages**: DDL&DML

**Level of abstraction**: DB -> physical schema-> logical schema -> view(multiple)

* **View**: how a user **sees** the data
* **Logical schema**: the **structure of data** shared among all users
* **Physical schema**: the **files and indexes** used for storage on disk

Data Independence

• Applications are insulated from how data is structured and stored

• Logical data independence: Protection from changes in logical schema (eg introducing an extra column in a table)

• Physical data independence: Protection from changes in physical structure and location of data

• Data independence is one of the most important benefits of using a DBMS

**Roles&career**s

Roles with DBMS

• End users- people who do something that advances the organization’ s purpose (business manager)

• DB application programmers- produce the applications that end users can run

• Database administrator (DBA)- manage effective & efficient use of resources in providing access to data

• DBMS Vendor Staff- DBMS software’s support provider from mother company

**Files vs DBMS**

**Files**: accessed directly by all the programs that need to use the data.

Pros: good for storing data for long time; easy to send and share

Cons: **low Integrity**(possible dup data); many **programs** needed(formats) ; assess control is mediocre; no central authority(no unified organize) ; Atomicity of updates(half-complete action cause error);

**DBMS**:

Pros: solve all cons above.

Cons: simple processing performance; money cost; handle specialised data poorly.

Week 2

**Outline**

- Introduction to SQL

- **Joins** and **Aggregate** Functions, **Set** operations

- Conceptual Database Design using the **Entity Relationship Model**

Introduction to SQL

Select

* SELECT a FROM b WHERE c ORDER by d ASC/DESC 正常选择
* SELECT DISTINCT/ALL a FROM b 选择全部/不同的行
* SELECT a, b, 2\*c(c is numerical) FROM d 多重选择， 选择时运算
* SELECT a AS xxx 选择后重命名列

Where

* WHERE a < 0 AND b = 0 筛选条件
* WHERE a BETWEEN 2 AND 4 区间内筛选条件

String operation

* ‘%’ match any substring %代表一切子字符串 eg comp%
* ‘\_’ match any char \_代表一切字符 eg comp2\_17

Date and Time

* EXTRACT component FROM date 从日期提取元素 eg EXTRACT year FROM enroldate
* TO\_DATE (string, templete) 将字符串变为日期 eg TO\_DATE(’01-03-2012’, ‘DD-MM-YYYY’)
* +/- INTERVAL 加减日期区间 eg ‘2012-04-01’ + INTERVAL ’36 HOUR’

Aliases

* SELECT A.name, M.name FROM Academic A, Academic M WHERE A.manager = M.empid

列出所有科研者以及他对应上级的名字

**JOIN function**

SQL join operators

* R natural join S
* R inner join S on <join condition>
* R inner join S using (<list of attributes>)

Eg List all students and in which courses they enrolled.

* select name, uos\_code, semester

from Student natural join Enrolled

eg Who is teaching “ISYS2120”?

* select name

from UnitOfStudy inner join Academic on lecturer=empid

where uos\_code=‘ISYS2120

**Aggregate function**

These functions operate on the multiset of values of a column of a relation, and return a value

* avg: average value
* min: minimum value
* max: maximum value
* sum: sum of values
* count: number of values

Must use **distinct** in addition to aggregate over sets, all these function **ignore NULL except ‘count’**.

**Conceptual database**

Database design sequence:

* requirement analysis: what data, what application, what operation is needed

conceptual design:

* develop high level description of data that match user need.

Logical design:

* convert conceptual design into logical database schema.

Physical design:

* convert logical schema into physical database.

Conceptual data model

Goal: Specification of database schema

Methodology:

* **Conceptual Design**: A technique for understanding and capturing business information requirements **graphically**
* Conceptual Database Design does **not** imply how data is implemented, created, modified, used, or deleted.
* An conceptual data model is model & database independent

Entity Relationship Model

Entity: object that store data

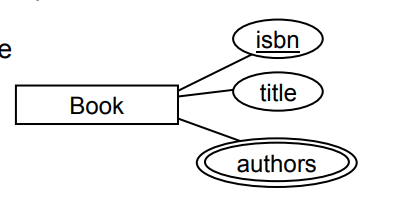
Entity type: collection of entity that share common attributes

Attribute: describe one property of an entity type

Domain: possible value of an attribute( 0 or 1, 1 – 10)

Key: minimal set of attributes that uniquely identifies an entity in the set.(primary key)

符号： 方块代表entity type， 椭圆代表attribute， 两圈代表多值属性， 下划线代表key



Relationship: relates two or more entities(number of entity = degree of relationship)

Relationship Type: set of similar relationships

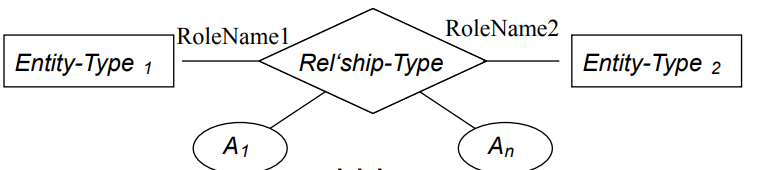
Relationship attribute:

* E.g., John enrols in ISYS2120 in the Second Semester 2018
* **2018 sec sem** is the attribute of relation **enrol**.

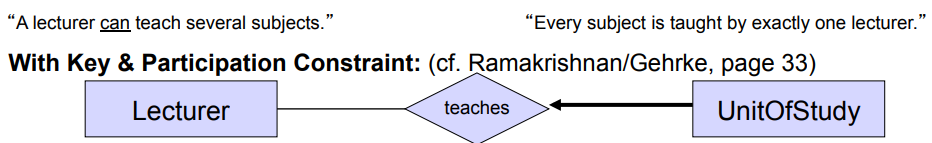
Relationship role: Each participating entity can be named with an explicit role.

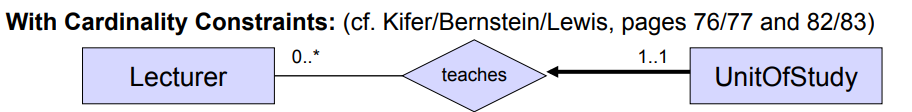
* E.g. John is value of Student role, ISYS2120 value of Subject role

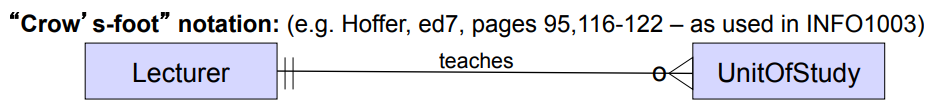
符号：菱形代表relationship type, relationship 也可以带有attribute。



Multiplicity





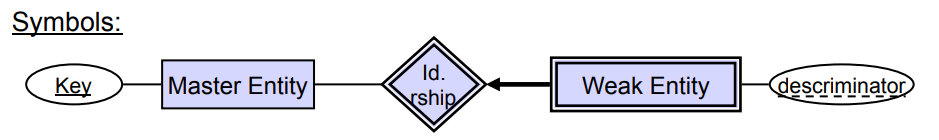


Weak entities

Weak entity type: An entity type that does not have a primary key.

* Can be seen as an exclusive ‘part-of’ relationship
* Its existence depends on the existence of one or more identifying entity types
* it must relate to the identifying entity set via a total, one-to-many identifying relationship type from the identifying to the weak entity set
* Examples: child from parents, payment of a loan

The **discriminator** (or **partial key**) of a weak entity type is the set of attributes that distinguishes among all the entities of a weak entity type related to the same owning entity.



Week 3: The Relational Data Model

**Outline:**

- Introduction to the **Relational Data Model** (关系数据模型)

- **Creating Relational Database Schemas using SQL**

- **Mapping E-R Diagrams to Relational Schemas**

**Data Model vs. Schema**

- **Data model**: a collection of concepts for describing:

- **Structure** of the data

- **Operations** on the data

- **Constraints** on the data

- **Relational data model** Main concept:

- **relation**, basically a table with rows and columns

- Every relation has a **schema**, which describes the columns, or fields

- **relation**: a named, two-dimensional **table** of data, consists of **rows (record, tuple)** and **columns (attribute or field)**

**- cardinality**: number of rows

- **degree/arity**: number of columns

**Relation requirement**:

- Every relation must have a unique name.

- Attributes (columns) in tables must have unique names. => The order of the columns is irrelevant.

- All tuples in a relation have the same structure; constructed from the same set of attributes

- Every attribute value is atomic (not multivalued, not composite).

- **Every row(tuple) is unique** (cant have two rows with exactly the same values for all their fields)

- The order of the rows is immaterial

- this is called **First Normal Form( 1NF)**

**Creating relational database schemas using SQL**

! Creation of tables (relations):

CREATE TABLE name ( list-of-columns )

* Eg: CREATE TABLE Instructor (lname VARCHAR(20),

fname VARCHAR(20),

salary INTEGER,

birth DATE,

hired DATE );

! Deletion of tables (relations):

DROP TABLE name

! Existing schemas can be changed

ALTER TABLE name ADD COLUMN … | ADD CONSTRAINT…| …

**Modifying Instances using SQL**

! **Insertion** of new data into a table / relation

! **Syntax**: INSERT INTO table [“(”list-of-columns“)”] VALUES “(“ list-of-expression “)”

! Example: INSERT INTO Student (sid, name) VALUES (12345678, ‘Smith’)

! **Updating** of tuples in a table / relation

! **Syntax**: UPDATE table

SET column“=“expression {“,”column“=“expression}

[ WHERE search\_condition ]

! Example: UPDATE Student

SET address = ‘4711 Water Street’

WHERE sid = 123456789

! **Deleting** of tuples from a table / relation

! **Syntax**: DELETE FROM table [ WHERE search\_condition ]

! Example: DELETE FROM Student WHERE name = ‘Smith’

**Integrity Constraints**

! **Integrity Constraint (IC)**: condition that **must be true** for **any instance** of the database; e.g., domain constraints.

- **Non-Null Columns**: value in a given column can’t be null

! **syntax**: CREATE TABLE Instructor

( lname VARCHAR(20) **NOT NULL**,

fname VARCHAR(20) **NOT NULL**);

! **Primary keys**: unique, minimal identifiers in a relation.

! **Foreign keys:** identifiers that enable a **dependent relation** (on the many side of a relationship) to refer to its **parent relation** (on the one side of the relationship), **must refer to a candicate key** in parental relation.

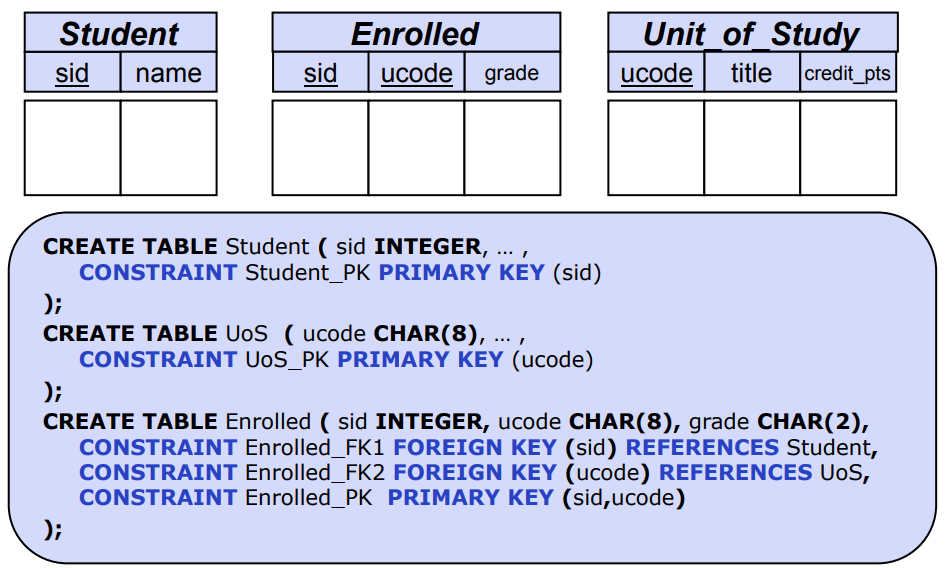
! A set of fields is **a key if**:

1. **No two distinct** tuples can **have same value**s in all key fields
2. This is not true for any subset of the key.( superkey if not)

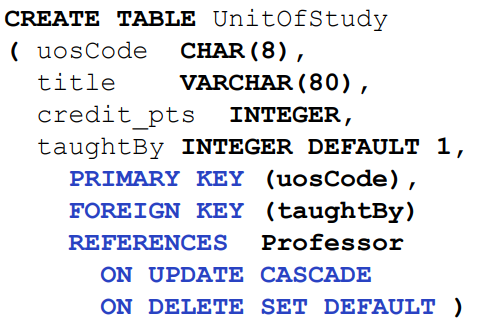
All those keys are called **candicate key, choose one as primary key**.

! Keys can be **simple** (single attribute) or **composite** (multiple attributes)

**Key constraints in SQL**



* Use **UNIQUE** to represent candidate key( also only once)
* When a tuple that’s refered by foreign key changes:
  + Default is **NO ACTION** (delete/ update is rejected)
  + **CASCADE** (also delete all tuples that refer to deleted tuple)
  + **SET NULL / SET DEFAULT** (sets foreign key value of referencing tuple)



**Mapping E-R Models to Relations**

Correspondence with E-R Model

! **Relations (tables)** correspond to **entity types (entity sets)** and to **many-to-many relationship types/sets**.

! **Rows** correspond with **entities** and with **many-to-many relationships**.

! **Columns** correspond with **attributes** or **one-to-many relationships.**

! Note: The word relation (in relational database) is **NOT the same** as the word relationship (in E-R model).

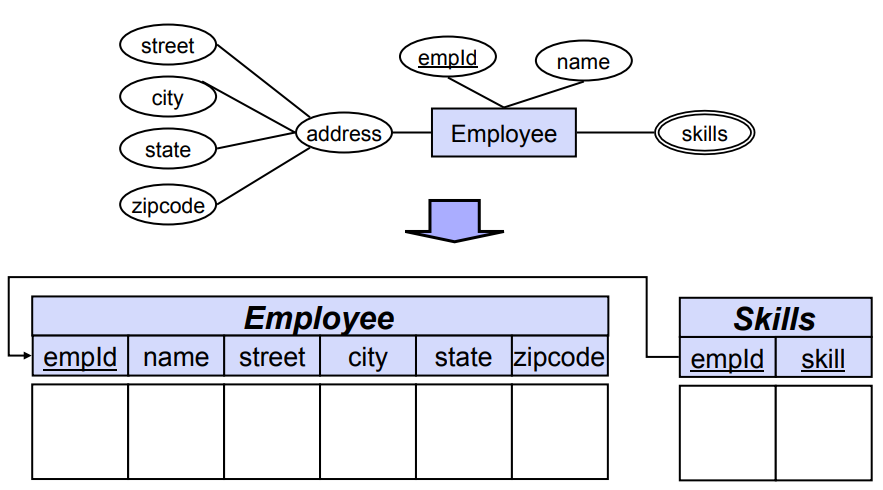
1. Mapping **Regular Entities** to **Relations**

! Each **entity type** becomes a **relation(table)**

! **Simple attributes**: E-R attributes map directly onto the relation

! **Composite attributes**: Composite attributes are flattened out by creating a separate field for each component attribute => We use only their simple, component attributes

! **Multi-valued attribute**: Becomes a separate relation with a foreign key taken from the superior entity

Eg. 

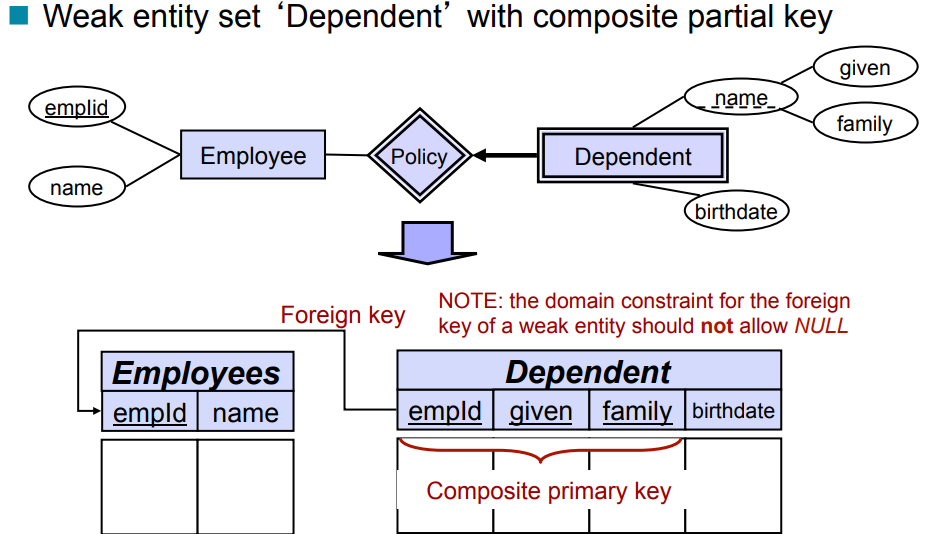
1. Mapping of **Weak Entity** Types

* become a separate relation with a foreign key taken from the superior entity
* primary key composed of:

- Partial key (discriminator) of weak entity

- Primary key of identifying relation (strong entity)

- Mapping of attributes of weak entity as shown before

-eg. 

1. Mapping of **Relationship Types**

! **Many-to-Many** - Create a new relation(table) with the primary keys of the two entity types as its primary key

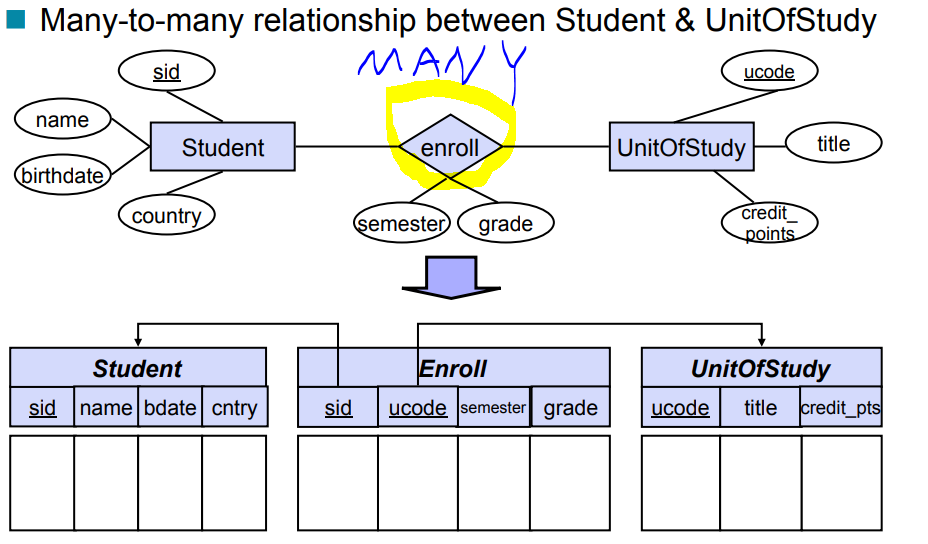
! **One-to-Many** - Primary key on the **one** side becomes a foreign key on the **many** side

! Participation Constraint: total side becomes **NOT NULL**

! **One-to-One** - Primary key on the **mandatory** side becomes a foreign key on the **optional** side

! **Relationship attributes** - become fields of either the dependent, respectively new relation

Eg. MANY-TO-MANY



Eg. ONE-TO-MANY

