

Databases Management Systems (DBMS)

1

Outline

- System outline (DBMS)
- Database design overview
- Entity-Relationship (ER) diagrams
- Relational data model
- Structured Query Language (SQL)
- Normalization
- ... and a few other issues

2

What is a DBMS ?

■ Database

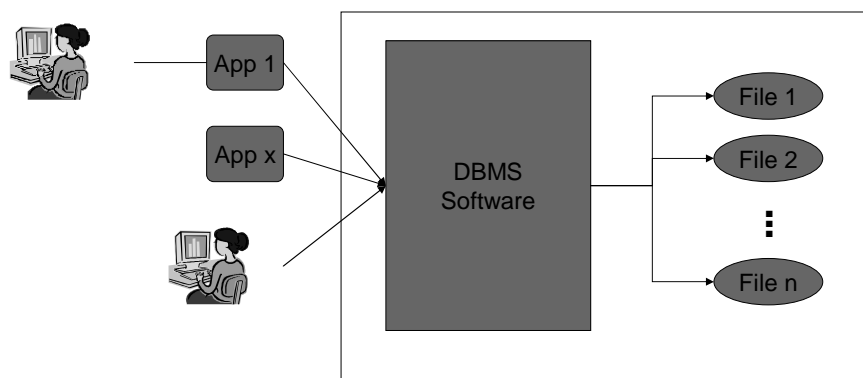
- a large, integrated collection of data
- Models a real-world enterprise
 - Entities (e.g. students, courses, instructors)
 - Attributes (e.g. student: ID, age, etc)
 - Relationships (e.g. Trappenberg is teaching ECMM 6010)

■ Database Management System (DBMS)

- a software application to assist in the creation, maintenance, and access of databases

3

DBMS



4

DBMS Features

- Integrated data store
 - Reduces data redundancy and inconsistency
- Query language
 - Reduces application development time
- Data access methods and query optimization
 - Ensures fast query answering for large data
- Data independence
 - application programs are not impacted by changes in the way the data is structured and stored
- Uniform data administration
 - Provides easy data integrity and security
- Concurrent control and crash recovery

5

Example – A Banking System

- Data = info on accounts, customers, balances, current interest rates, transaction histories, etc.
- Features:
 - **Massive:** many terabytes at least for large banks
 - **Shared:** reduce redundancy and ensure data consistency and control of access
 - **Persistent:** data outlives programs that operate on them
 - **Safe:** from system failures (leveraged through software and hardware)
 - **Multi-user:** many people / programs accessing same database simultaneously
 - **Convenient:** simple commands, data independences facilitates application development
 - **Efficient:** don't search all files

6

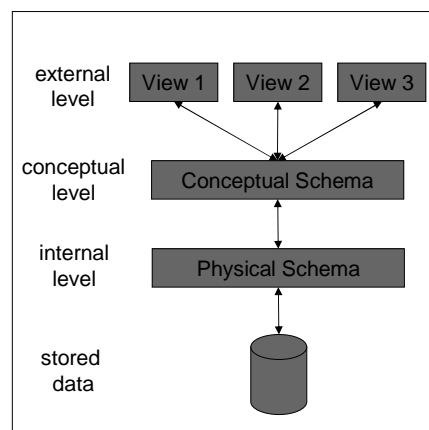
Describing and Storing Data

- Relationship between real-world enterprise and data
- Data Model
 - a mechanism for describing data that hides the low-level storage details
 - Relational, network, hierarchical, object-oriented, etc.
- Relational data model
 - most widely used
 - a collection of relations (set of records) and operations on relations

7

Levels of abstraction

- External level
 - many views
 - describes how the users see the data
- Conceptual (logical) level
 - defines what is in the database
- Internal level
 - describes how the data is actually stored
- Stored data



8

Example: University Database

- External schema:
 - View - Courseinfo (*cid:string, enrollment:integer*)
- Conceptual schema:
 - Students (*sid:string, name:string, age:integer, gpa:real, major:string*)
 - Courses (*cid:string, cname:string, credits:integer*)
 - Coursereg (*sid:string, cid:string, grade:string*)
- Physical schema:
 - relations stored as unordered files, indexed on first column of students, etc.

9

Data Independence

- Applications are protected from how data is structured and stored
- Logical data independence
 - protection from changes in logical level of data
 - provided by the external and conceptual levels. If the conceptual schema changes, the views can be reconfigured to retrieve the data using the new conceptual schema
- physical data independence
 - the code remains the same is the underlying data structure is changes

10

Database Design (1)

1. Requirements analysis
 - understand what data is to be stored in the database, what applications must be built on top of it, and what operations will be most frequent
 - i.e. find out what the users want!
2. Conceptual database design
 - use the info gathered in step 1 to develop a high-level description of the data to be stored, along with the constraints that are known
 - entity-relationship model often used for this step
3. Logical database design
 - choose a DBMS to implement the database design and convert the conceptual database design into a conceptual schema (which can be implemented, for example, with SQL)

11

Database Design (2)

4. Schema refinement
 - analyze the relations to identify potential problems and refine the design (normalization)
5. Physical database design
 - consider typical expected workloads and refine the database design to meet performance criteria
 - i.e. building indexes or potentially redesign
6. Security design
 - identify different user groups and roles and identify access restrictions

12

Database Design Step 2

1. Requirements analysis
- 2. Conceptual database design**
 - use the info gathered in step 1 to develop a high-level description of the data to be stored, along with the constraints that are known
 - entity-relationship model often used for this step
3. Logical database design
4. Schema refinement
5. Physical database design
6. Security design

13

Entity-Relationship (ER) Model

- Conceptual Database Design
- Standard technique for describing data in terms of objects and their relationships
- Entity
 - an object in the real world that is distinguishable from other objects (i.e. student, instructor, course, textbook)
 - an entity is described used a set of attributes (i.e. Sid, Name, Sex, dob)
- Entity set
 - a collection of similar entities (i.e. students, instructors)
 - all entities in an entity set have the same attributes (for now, we will expand on this later)
 - each entity has a **key** (a minimal set of attributes that uniquely identifies an entity) & a **domain** (a set of possible values for the attributes)

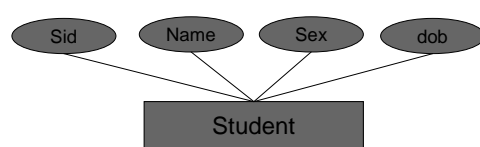
14

ER Diagrams

 Entity set

 Attribute

 Relationship



Sid	Name	Sex	dob
1234	Thomas	M	01/01/85
2345	Quinn	F	05/05/68
5532	Smith	M	10/10/75

15

Type of Attributes

- Simple (atomic) vs. composite
 - (e.g. house number vs. address)
- Single-valued vs. multi-valued
 - (e.g. age versus university degrees)
- Stored vs. derived
 - (e.g. birth date versus age)

16

Relationships

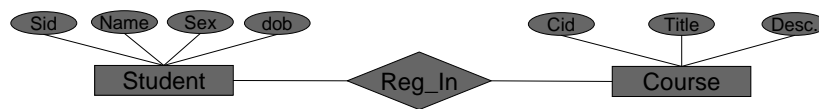
- Relationship

- an association among two or more entities (i.e. Thomas is registered in ECMM6010)

- Relationship set

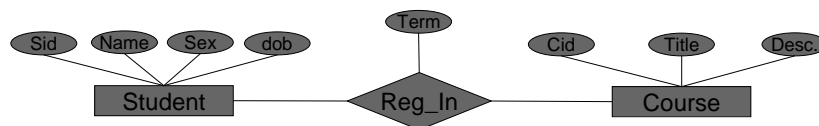
- a collection of similar relationships (i.e. Reg_In)
- can also have descriptive attributes (but the relationship must be uniquely identified by the participating entities)

Sid	Cid
1234	ECMM6010
2345	CSCI4161
5532	ECMM6000



17

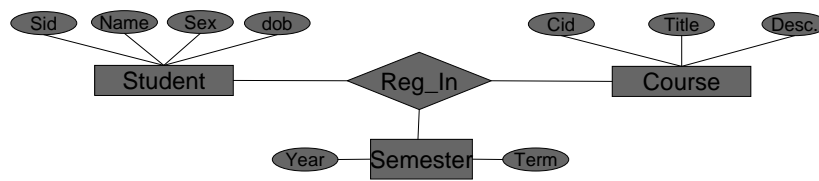
Ternary Relationships



- Sometimes there is a three-way relationship (i.e. student / course / semester)
- The above diagram is ONLY appropriate if any student only takes a course once (i.e. only one term associated with that student/course pair)

18

Ternary Relationships



- If a student can take a course several times, you need to represent a third entity "Semester" in order to represent this relationship

19

Key attribute (key/uniqueness constraint)

- Key attribute(s):
 - Attribute(s) that uniquely defines individual entity
- Key constrain:
 - there can be only one instance of an item in a relationship (i.e. each department can only have one manager)
 - represented in an ER diagram by adding an arrowhead to the line between the entity and the relationship



20

Relationship Constraints

Cardinality ratio (N:M)

one-to-one



one-to-many





many-to-one



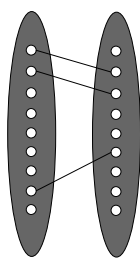
many-to-many



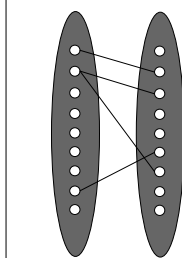
Participation Constraints: partial 
total 

21

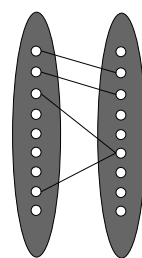
Cardinality ratio (N:M)



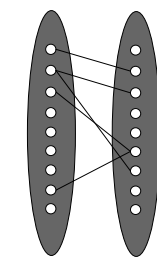
one-to-one



one-to-many



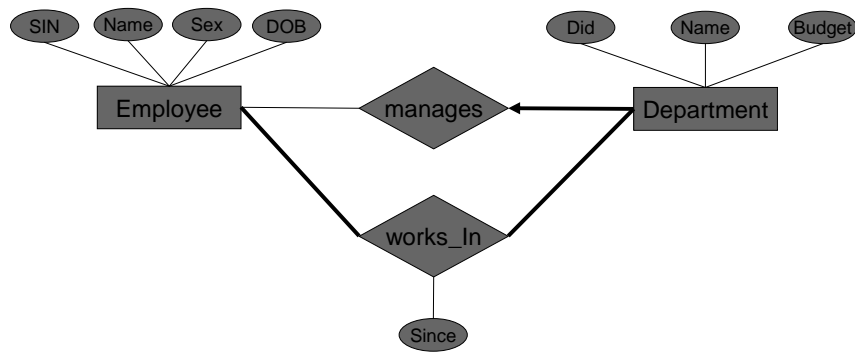
many-to-one



many-to-many

22

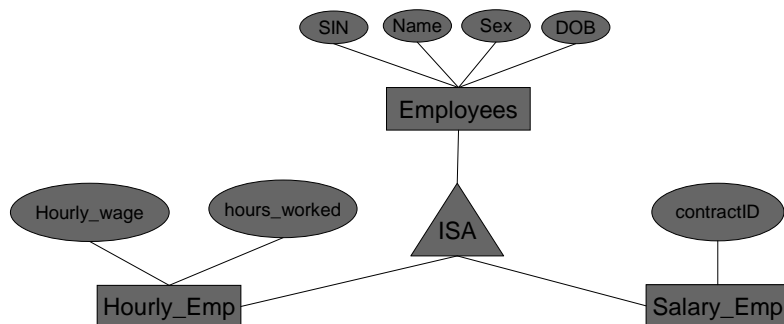
Example – Employees & Departments



23

ISA ('is a') Hierarchies

- Attributes can be inherited
- If we declare A ISA B, then every A entity is also considered to be a B entity



24

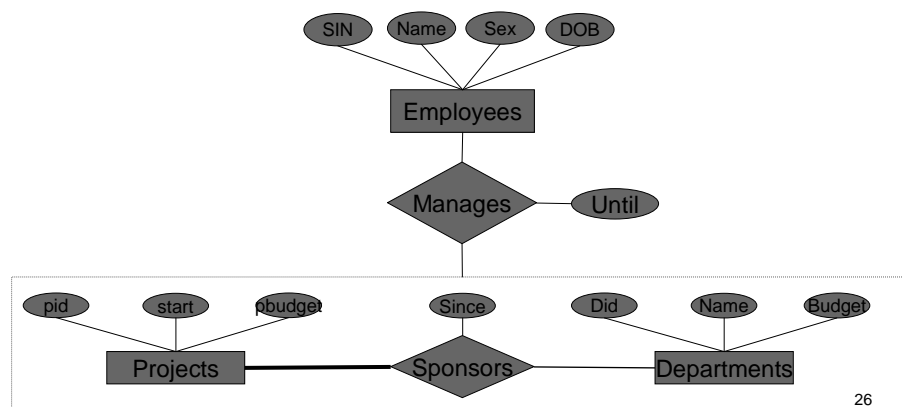
ISA Hierarchy Constraints

- **Overlap constraints**
 - are two subclasses allowed to contain the same entity? (i.e. can Joe be an hourly and a salary employee?)
 - If yes we write “Hourly_Emp OVERLAPS Salary_Emp”
- **Covering constraints**
 - do the entities from the subclasses comprise all entities from the superclass? (i.e. does every employee have to be an hourly or a salary employess?)
 - If yes we write “Hourly_Emp AND Salary_Emp COVER Employees”

25

Aggregation

- enables a relationship to be treated as an entity set for the purpose of participating in other relationships
- represented by a dashed box around the relationship



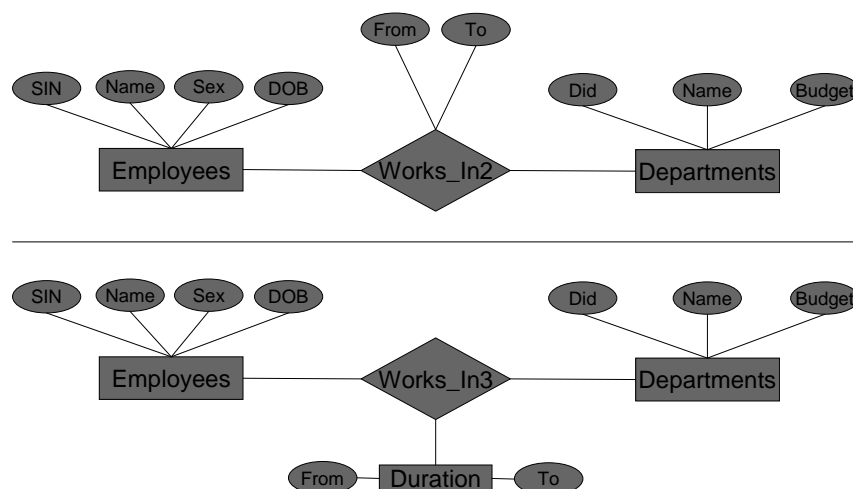
26

Design choice: entity vs. attribute

- Should a property be modeled as an attribute or an entity? (i.e. address)
- Depends on what we want to do with the information
- Should be an entity if:
 - we want to record more than one (i.e. more than one address for each employee)
 - we want to capture the structure of the attribute (i.e. want to record address in terms of street, city, province, country, postal code. This was we can query on this info)

27

Design choices: entity vs. attribute



28

Summary of conceptual design

- conceptual design follows requirements analysis
 - gives a high-level description of data to be stored
- ER model frequently used
- Basic components: entities, relationships, and attributes (or entities and relationships)
- Additional components: ISA hierarchies
- Can express constraints (i.e. key constraints, participation constraints, overlap/covering constraints)
- ER design is subjective. There are often many ways to model a scenario!

29

Database Design Step 3

1. Requirements analysis
2. Conceptual database design
- 3. Logical database design**
 - choose a DBMS to implement the database design and convert the conceptual database design into a conceptual schema (which can be implemented, for example, with SQL)
4. Schema refinement
5. Physical database design
6. Security design

30

Logical database design

- convert the conceptual database design into a conceptual schema
- ER Model → Relational Model

- Relational data model
 - based on sets
 - tabular representation of data
 - a relation consists of an instance (table) with a schema (column headings)

Reminder: a schema is a description of data in terms of a data model

31

Relation schema

- specifies the relation's name, the name of each field (or column, or attribute), and the domain of each field
- domain has a domain name and a set of associated values

Students (Sid: integer, Sname: string, gpa: real)

Course (Cid: integer, Cname: string, Ctitle: string)

Takes (Sid: integer, Cid: integer)

32

Relation instance

- set of records (also called tuples)
- each record has the same number of fields as the relation schema
- basically a table where each row represents a record

FIELDS (ATTRIBUTES, COLUMNS)

Sid	Name	gpa
1234	Thomas	0.07
2345	Quinn	4.03
6672	Jones	3.10
5532	Smith	2.17

RECORDS (TUPLES, ROWS)

An instance of the Students relation

- all rows in a table must be distinct
- cardinality is the number of records (tuples); above: cardinality = 4
- degree is the number of fields (columns); above: degree = 3

33

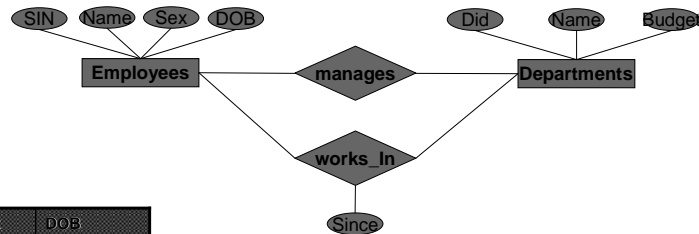
Table Design Phase

- Rules of thumb
 - one table per entity
 - each attribute of an entity becomes an attribute of the table
 - one table per relationship
 - each attribute of a relationship becomes an attribute of the table
 - the primary keys for the participating entity sets become part of the table (if no key constraints)

34

Example – Employees & Departments

ER Diagram



Tables

Employees

SIN	Name	Sex	DOB
1234	Thomas	M	01/01/85
2345	Quinn	M	05/05/68
6672	Jones	M	11/11/72
5532	Smith	F	06/18/43

Departments

Did	Name	Budget
23	Dept A	500000
18	Dept B	200000
33	Dept C	750000

manages

SIN	Did
1234	23
2345	33

works_In

SIN	Did	Since
1234	23	1990
2345	33	1982
6672	23	2000
5532	18	2002

35

Relational Rules

- There is no meaning to the order of tuples in a relation
- There is no meaning in the order of attributes in a relation
- Each tuple is unique in a relation
- Each attribute value is atomic (i.e. cannot be split up)
- Each tuple is an assertion (i.e. a true statement)

36

Integrity Constraints

- Data entered into a database must be correct and a DBMS should help prevent incorrect information
- An integrity constraint is a condition that is specified on a database schema that restricts what data can be stored
- Types of integrity constraints include:
 - keys
 - foreign key constraints
 - domain constraints
 - participation constraints

37

Keys

- A set of fields is a *key* for a relation if:
 - uniqueness: no two distinct tuples can have the same values in all key fields (key can be a combination of attributes)
 - minimality: not true for any subset of the key
- If minimality is false it is considered to be a *superkey*
 - *set of all attributes is a superkey*
 - *Key = minimal superkey*
- There may be several possible keys (*candidate keys*)
- The *primary key* is the key chosen to be supported in the database
- A primary key CANNOT be null

38

Primary Key

- Primary key
 - attribute(s) that uniquely identify records (tuples)
 - can be comprised of one or more attributes

Students (**Sid**: integer, Sname: string, gpa: real)

Course (**Cid**: integer, Cname: string, Ctitle: string)

Takes (**Sid**: integer, **Cid**: integer)

39

Primary Key

Employees

<u>SIN</u>	Name	Sex	DOB
1234	Thomas	M	01/01/85
2345	Quinn	M	05/05/68
6672	Jones	M	11/11/72
5532	Smith	F	06/18/43

Departments

<u>Did</u>	Name	Budget
23	Dept A	500000
18	Dept B	200000
33	Dept C	750000

Manages

<u>SIN</u>	<u>Did</u>
1234	23
2345	33

Works_In

<u>SIN</u>	<u>Did</u>	Since
1234	23	1970
2345	33	1982
6672	23	2000
5532	18	2002

40

Foreign Key

- Attribute(s) that link to a primary key in another relation
 - For example: in the relation Reg_in (sid: int, cid: int, grade: int), sid would be a foreign key (from the Students relation) and cid would be a foreign key (from the courses relation)
- Referential integrity
 - If a foreign key refers to a primary key in another relation, then that key must exist in the other relation

41

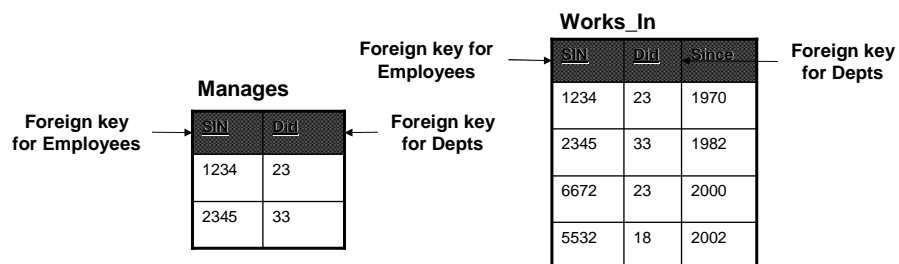
Foreign Key

Employees

<u>Sid</u>	Name	Sex	DOB
1234	Thomas	M	01/01/85
2345	Quinn	M	05/05/68
6672	Jones	M	11/11/72
5532	Smith	F	06/18/43

Departments

<u>Did</u>	Name	Budget
23	Dept A	500000
18	Dept B	200000
33	Dept C	750000



42

Translating Key Constraints (1)



■ Option #1

- create a table for the relation, and the primary key for that table will be the primary key from Department (did)

Manages

<u>SIN</u>	<u>Did</u>
1234	23
2345	33

43

Translating Key Constraints (2)

■ Option #2

- include information about the relationship in the table corresponding to the Department entity
- Advantage: don't need to create an extra table or join tables for queries
- Disadvantage: can be a waste of space if several departments don't have managers

Departments

<u>Did</u>	<u>Name</u>	<u>Budget</u>	<u>Manager</u>
23	Dept A	500000	1234
18	Dept B	200000	
33	Dept C	750000	2345

44

Translating Participation Constraints



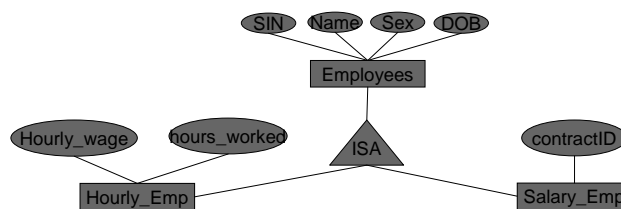
- If the participation constraint is associated with a key constraint we can specify "NOT NULL" to constrain total participation IF design option #2 is used
- This is not the case if option #1 was used to handle the key constraint
- For other participation constraints, table constraints or assertions must be used

Departments

Djd	Name	Budget	Manager (NOT NULL)
23	Dept A	500000	1234
18	Dept B	200000	6672
33	Dept C	750000	2345

45

Translating ISA Class Hierarchies (1)



- Option #1
 - create a table for each entity set (employees, hourly_emp, and salary_emp)
 - the subclasses (hourly_emp and salary_emp) contain the key attribute from the superclass (employees)
 - good if we run a lot of queries on employees in general
 - requires joining two tables though if we want to compare just hourly employees

46

ISA Class Hierarchy Example

Option #1

Employees

SIN	Name	Sex	DOB
1234	Thomas	M	01/01/85
2345	Quinn	M	05/05/68
6672	Jones	M	11/11/72
5532	Smith	F	06/18/43

Hourly_Emp

SIN	Hourly_w age	Hours_w orked
1234	7.00	27
2345	6.50	34
6672	12.00	37
5532	11.00	55

SIN	Contract ID
1234	55
2345	73
6672	120
5532	26

47

Translating ISA Class Hierarchies (2)

■ Option #2

- create two tables, one for each subclass (hourly_emp and salary_emp) and include all of the attributes from the employee class in each table (employees, hourly_emp, and salary_emp)
- must join tables to compare employees in general but is faster to do comparisons within each category
- doesn't work if we do not have a covering constraint (i.e. have employees that are not either an hourly employee or a salary employee)
- also, redundancy if we have an overlap constraint (i.e. an employee is both an hourly and a salary employee)

48

ISA Class Hierarchy Example

Option #2

Hourly_Emp

SIN	Name	Sex	DOB	Hourly_wage	Hours_worked
1234	Thomas	M	01/01/85	7.00	27
2345	Quinn	M	05/05/68	6.50	34
6672	Jones	M	11/11/72	12.00	37
5532	Smith	F	06/18/43	11.00	55

Salary_Emp

SIN	Name	Sex	DOB	Contract ID
1234	Thomas	M	01/01/85	55
2345	Quinn	M	05/05/68	73
6672	Jones	M	11/11/72	120
5532	Smith	F	06/18/43	26

49

Other constraints

- Domain constraints
 - May specify a range of values for a field
- Table constraints
 - Constraints associated with a single table and are checked whenever that table is modified
- Assertions
 - Involve several tables and are checked whenever any of these tables are modified
- Overlap & covering constraints are handled using assertions

50

SQL (structured query language)

- The most widely used relational query language for creating, manipulating and querying relational databases

51

Creating Tables with SQL

```
CREATE TABLE Students (sid: INTEGER, name: CHAR(20), age:
    INTEGER, gpa: REAL)
```

```
CREATE TABLE Enrolled (sid: INTEGER, cid: INTEGER, grade:
    CHAR(2))
```

Students

Sid	Name	Age	Gpa
-----	------	-----	-----

Enrolled

Sid	Cid	Grade
-----	-----	-------

52

Adding Keys with SQL

```
CREATE TABLE Students (sid: INTEGER, name: CHAR(20), age:
    INTEGER, gpa: REAL, PRIMARY KEY (sid))
```

```
CREATE TABLE Enrolled (sid: INTEGER, cid: INTEGER, grade:
    CHAR(2), PRIMARY KEY (sid, cid), FOREIGN KEY (sid)
    REFERENCES Students, FOREIGN KEY (cid) REFERENCES
    courses)
```

Students

Sid	Name	Age	Gpa
-----	------	-----	-----

Enrolled

Sid	Cid	Grade
-----	-----	-------

53

Inserting & Updating Data

```
INSERT INTO Students (sid, name, age, gpa) VALUES
    (1234, 'Thomas', 18, 4.1)
```

```
UPDATE Students SET Name = Johnson WHERE sid = 5532
```

Sid	Name	Age	GPA
1234	Thomas	18	4.1
2345	Quinn	32	3.6
5532	Johnson	99	3.9

- Often DBMS provide a GUI to facilitate data entry/deletion/modification or provides facilities for the designer to create a form for data entry/deletion/modification

54

Querying with SQL (data retrieval)

- **SELECT:** Choose one or more rows

```
SELECT * FROM students WHERE name = "Thomas"
```

- **PROJECT:** Choose one or more columns:

```
SELECT sid, name FROM students
```

- **JOIN:** combine two tables to make a new one

```
SELECT * FROM students, enrolled WHERE students.sid =  
enrolled.sid
```

55

SELECT Example 1

- Choose rows from a table

Employees

SIN	Name	Sex	DOB
1234	Thomas	F	01/01/85
2345	Quinn	M	05/05/68
6672	Jones	M	11/11/72
5532	Smith	F	06/18/43

```
SELECT * FROM Employees WHERE Sex = "M"
```

RESULT

SIN	Name	Sex	DOB
2345	Quinn	M	05/05/68
6672	Jones	M	11/11/72

56

SELECT Example 2

- Choose rows (and specific fields) from a table

SIN	Name	Sex	DOB
1234	Thomas	F	01/01/69
2345	Quinn	M	05/05/68
6672	Jones	M	11/11/72
5532	Smith	F	06/18/43

```
SELECT SIN, Name FROM Employees WHERE Sex = "M"
```

RESULT

SIN	Name
2345	Quinn
6672	Jones

57

PROJECT Example 1

- Choose one or more columns from a table

SIN	Name	Sex	DOB
1234	Thomas	M	01/01/85
2345	Quinn	M	05/05/68
6672	Jones	M	11/11/72
5532	Smith	F	06/18/43

```
SELECT Name, DOB FROM Employees
```

RESULT

Name	DOB
Thomas	01/01/85
Quinn	05/05/68
Jones	11/11/72
Smith	06/18/43

58

PROJECT Example 2

- Choose one or more columns (but only for records that satisfy a constraint) from a table

Employees

SIN	Name	Sex	DOB
1234	Thomas	F	01/01/85
2345	Quinn	M	05/05/68
6672	Jones	M	11/11/72
5532	Smith	F	06/18/43

```
SELECT Name FROM Employees WHERE SIN > 3000
```

RESULT

Name
Jones
Smith

59

JOIN Example

- gather information that is stored in two different tables

Students

Sid	Name	Age	GPA
1234	Thomas	18	4.1
2345	Quinn	34	3.6
6672	Jones	30	0.07
5532	Smith	99	3.9

Enrolled

Sid	Cid	Grade
1234	63	A
5532	32	D
2345	81	B

```
SELECT * FROM Students, Enrolled
WHERE Students.Sid = Enrolled.Sid
```

RESULT

Sid	Name	Age	GPA	Cid	Grade
1234	Thomas	18	4.1	63	A
2345	Quinn	34	3.6	81	B
5532	Smith	99	3.9	32	D

60

Database Design Step 4

1. Requirements analysis
2. Conceptual database design
3. Logical database design
- 4. Schema refinement**
 - *analyze the relations to identify potential problems and refine the design (e.g. normalization)*
5. Physical database design
6. Security design

61

Schema Refinement

- Need to examine the database design, looking more closely at the issue of redundant storage of information
 - Problems caused by redundant storage
 - update anomalies
 - insertion anomalies
 - deletion anomalies

62

Problems with redundancy - example

- Suppose that hourly wages are determined by an employees rating
 - this is called a functional dependency

Hourly_Emp

SIN	Name	Sex	DOB	Rating	Hourly_wage	Hours_worked
1234	Thomas	M	01/01/85	8	10.00	27
2345	Quinn	M	05/05/68	5	6.50	34
6672	Jones	M	11/11/72	8	10.00	37
5532	Smith	F	06/18/43	5	6.50	55

63

Problems with redundancy - example

- Disadvantages:
 - information stored multiple times (wasted storage space)
 - update anomaly
 - the hourly wage for Thomas could be updated without updating the hourly wage for Jones which would produce an inconsistency
 - insertion anomaly
 - we cannot insert an employee unless we know the hourly wage for the employee's rating value
 - deletion anomaly
 - if we delete all tuples with a given rating value we lose the association between the rating value and its hourly wage

64

Solution – decomposition

- Decompose the larger relation into smaller relations
 - Hourly_emp2 (SIN, Name, Sex, DOB, rating, hours_worked)
 - Wages (rating, hourly_wages)
- Can update without creating inconsistencies and more efficient to update in one place
- Can insert an employee without knowing the hourly wage
- Can delete all employee of a certain rating without losing the information related to that rating and hourly wage

65

When do we need to decompose?

- Several standards have been identified which eliminate certain types of problems
 - normal forms (i.e. 1st, 2nd, 3rd, Boyce-Codd)
- Knowing the “normal form” of a schema, we can decide whether or not to decompose further
- What are the problems associated with decomposition?

66

1st normal form

- if every field contains only single values (i.e. no multi-valued or composite attributes)

Example (NOT IN 1st normal form):

STUDENT (snum, sname, {phone1, phone2}, address)

Correction (IN 1st normal form):

STUDENT (snum, sname, address)

PHONE (snum, phone1, phone2)

- By definition, all relational schemas are in 1st normal form

67

2nd normal form

- all attributes are fully dependent on the primary key

Example (NOT IN 2nd normal form):

COURSE (cnum, csec, room, cname)

Correction (IN 2nd normal form):

COURSE (cnum, csec, room)

COURSE_NAME (cnum, cname)

68

2nd normal form

Violation of 2nd normal form

Course

<u>Cid</u>	<u>Csec</u>	Room	Name
6000	01	333	Overview of E-Commerce
6010	01	258	Technology for E-Commerce
6010	54	319	Technology for E-Commerce
6030	01	107	Business for E-Commerce

Solution

Course

<u>Cid</u>	<u>Csec</u>	Room
6000	01	333
6010	01	258
6010	54	319
6030	01	107

Course_Name

<u>Cid</u>	Name
6000	Overview of E-Commerce
6010	Technology for E-Commerce
6030	Business for E-Commerce

69

3rd normal form

- no dependencies other than on the primary key

Example (NOT in 3rd normal form):

COURSE (cnum, csec, prof, prof_office)



Correction (IN 3rd normal form):

COURSE (cnum, csec, prof)

PROF_OFFICES (prof, prof_office)

70

3rd normal form

Violation of 3rd normal form

Course

<u>Cid</u>	<u>Csec</u>	Prof_id	Prof_office
6000	01	8703	CS 117
6010	01	7742	CS 333
6010	54	2251	CS 095
6030	01	8703	CS 117

Solution

Course

<u>Cid</u>	<u>Csec</u>	Prof_id
6000	01	8703
6010	01	7742
6010	54	2251
6030	01	8703

Prof_Office

Prof_id	Prof_office
8703	CS 117
7742	CS 333
2251	CS 095

71

What normal form is this example in?

- 1NF: if every field contains only single values
- 2NF: all attributes are fully dependent on the primary key
- 3NF: no dependencies other than on the primary key

Hourly_Emp

SIN	Name	Sex	DOB	Rating	Hourly_wage	Hours_worked
1234	Thomas	M	01/01/85	8	10.00	27
2345	Quinn	M	05/05/68	5	6.50	34
6672	Jones	M	11/11/72	8	10.00	37
5532	Smith	F	06/18/43	5	6.50	55

72

1st vs 2nd vs 3rd Normal Form

- A relation in 2nd normal form is also in 1st normal form
- A relation in 3rd normal form is also in 2nd and 1st normal form

73

And so on

- There are other forms of normalization but in most cases 3NF is good enough
- We need to evaluate the 'normal form' of a schema and decide if further decomposition is necessary

74

Database Design Step 5

1. Requirements analysis
2. Conceptual database design
3. Logical database design
4. Schema refinement
- 5. Physical database design**
 - *consider typical expected workloads and refine the database design to meet performance criteria (e.g. building indexes or schema refinement)*
6. Security design

75

Physical database design

- must address performance goals based on anticipated workload:
 - what queries will be most frequent?
 - what updates will be most frequent?
 - how fast certain queries or updates must run?
- Must identify
 - which tables must be accessed
 - which attributes are gathered
 - joins
 - type of updates (e.g. insert, delete, update)

76

Mechanisms to improve performance

- index creation
 - data storage to speed up retrieval
 - rules of thumb
 - only build indexes that are needed
 - index attributes mentioned in a WHERE clause

77

Mechanisms to improve performance

- denormalization
 - we may want to denormalize a schema in order to reduce the number of joins in frequent queries

Hourly_Emp

SIN	Name	Sex	DOB	Rating	Hourly_wage	Hours_worked
1234	Thomas	M	01/01/85	8	10.00	27
2345	Quinn	M	05/05/68	5	6.50	34
6672	Jones	M	11/11/72	8	10.00	37
5532	Smith	F	06/18/43	5	6.50	55

78

Database Design Step 6

1. Requirements analysis
2. Conceptual database design
3. Logical database design
4. Schema refinement
5. Physical database design
- 6. Security design**
 - *identify different user groups and roles and identify access restrictions*

79

Three Security Objectives

- **secrecy**
 - information should not be disclosed to unauthorized users
- **integrity**
 - only authorized users should be allowed to modify data
- **availability**
 - authorizing users should not be denied access

80

Transaction Processing

- A transaction is defined as a series of reads and writes of database objects” (Ramakrishnan, pg. 524)
- How a DBMS handles transactions is important for concurrency control and recovery

81

Concurrency Control Problem

User A	Joint Account	User B
Read total: \$100	\$100	
		Read total: \$100
Take out \$50		
		Take out \$50
Write total	\$50	
	\$50	Write total

This wouldn't be very good for the Bank!

82

ACID Transaction Model

- Four important properties of transactions:
 1. Atomic: each transaction is either carried out completely or not at all
 2. Consistent: each transaction must preserve consistency of the database
 3. Isolated: transactions are not affected by concurrent transactions
 4. Durable: once a transactions has been completed, its effects should persist, even if the system crashes

83

Transaction Concurrency

- Why do we want concurrent transactions?
- What are the problems with concurrent transactions?
- What is the simplest way to handle concurrent transactions?
- A schedule is a potential execution sequence for the actions in a set of transactions

84

Recovery

- DBMS is responsible for guaranteeing either
 - a transaction is completed successfully OR
 - a transaction has no effect on the data or any other transaction
- But, what about when a system crashes?
- A recovery manager ensures:
 - atomicity: must undo the actions of transactions that do not commit
 - durability: make sure that all the actions of a committed transaction survive system crashes
- The recovery manager maintains a log of all modifications to the database

85

References

- Raghu Ramakrishnan & Johannes Gehrke, *Database Management Systems*, McGraw Hill, 2000.
- Elmasri & Navathe, *Fundamentals of Database Systems*, Benjamin/Cummings, 2 Ed., 1994
- Darrel Ince, *Developing Distributed and E-commerce Applications*, Chapter 5, Addison Wesley 2002

86