Chapter 2:

Intro to Relational Model

Overview of Database Development

- *Requirements Analysis*
  - What data are to be stored in the enterprise?
  - What are the required applications?
  - What are the most important operations?
- *High-level database design*
  - What are the *entities* and *relationships* in the enterprise?
  - What information about these entities and relationships should we store in the database?
  - What are the *integrity constraints* or *business rules* that hold?
- *ER model to represent high-level design*

Overview of Database Development

- *Conceptual database design*
  - What data model to implement the DBS? E.g., relational data model
  - Map the high-level design (e.g., ER diagram) to a (conceptual) database schema of the chosen data model.
- *Physical database design*
  - What DBMS to use?
  - What are the typical workloads of the DBS?
  - Build indexes to support efficient query processing.
  - What redesign of the conceptual database schema is necessary from the point of view of efficient implementation?

Overview of Database Development

Requirements Analysis / Ideas

High-Level Database Design

Conceptual Database Design / Relational Database Schema

Physical Database Design / Relational DBMS

- Similar to software development

Relational Model

Why Study the Relational Model?

- Most widely used model by industry.
  - IBM, Informix, Microsoft, Oracle, Sybase, etc.
- It is simple, elegant, and efficient
  - Entities and relations are represented as tables
  - Tables allow for arbitrary referencing
    (Tables can refer to other tables)
- Forms a basis for other models
  - Object Oriented Model: ObjectStore, Versant, JSON
  - A synthesis emerging: *object-relational model*
    - Informix Universal Server, UniSQL, O2, Oracle, DB2
  - Column stores
  - Used in alternative implementations of RDF, XML

Relational Database: Definitions

- Relational database*:* a set of relations

- Relation: made up of 2 parts:
  - Instance : a table, with rows and columns.
    #rows = cardinality, #fields = degree / arity.

  - Schema : specifies name of relation, plus a name and type for each column.
    - e.g. Students(*sid*: string, *name*: string, *login*: string,
      age: integer, *gpa*: real).

- Can think of a relation as a *set* of rows or *tuples*.

Example of a Relation

attributes
(or columns)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

tuples
(or rows)

Attribute Types

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value ***null*** is a member of every domain
- The null value causes complications in the definition of many operations

Relation Schema and Instance

- $A_1, A_2, \ldots, A_n$ are *attributes*

- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*
- Example:
- *instructor* = (*ID, name, dept_name, salary*)
- Formally, given sets $D_1, D_2, \ldots. D_n$ a **relation** ***r*** is a subset of

  $D_1 \times D_2 \times \ldots \times D_n$

  Thus, a relation is a set of *n*-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$

- The current values (**relation instance**) of a relation are specified by a table
- An element ***t*** of ***r*** is a *tuple*, represented by a *row* in a table

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts

    instructor
    student
    advisor

- Bad design:

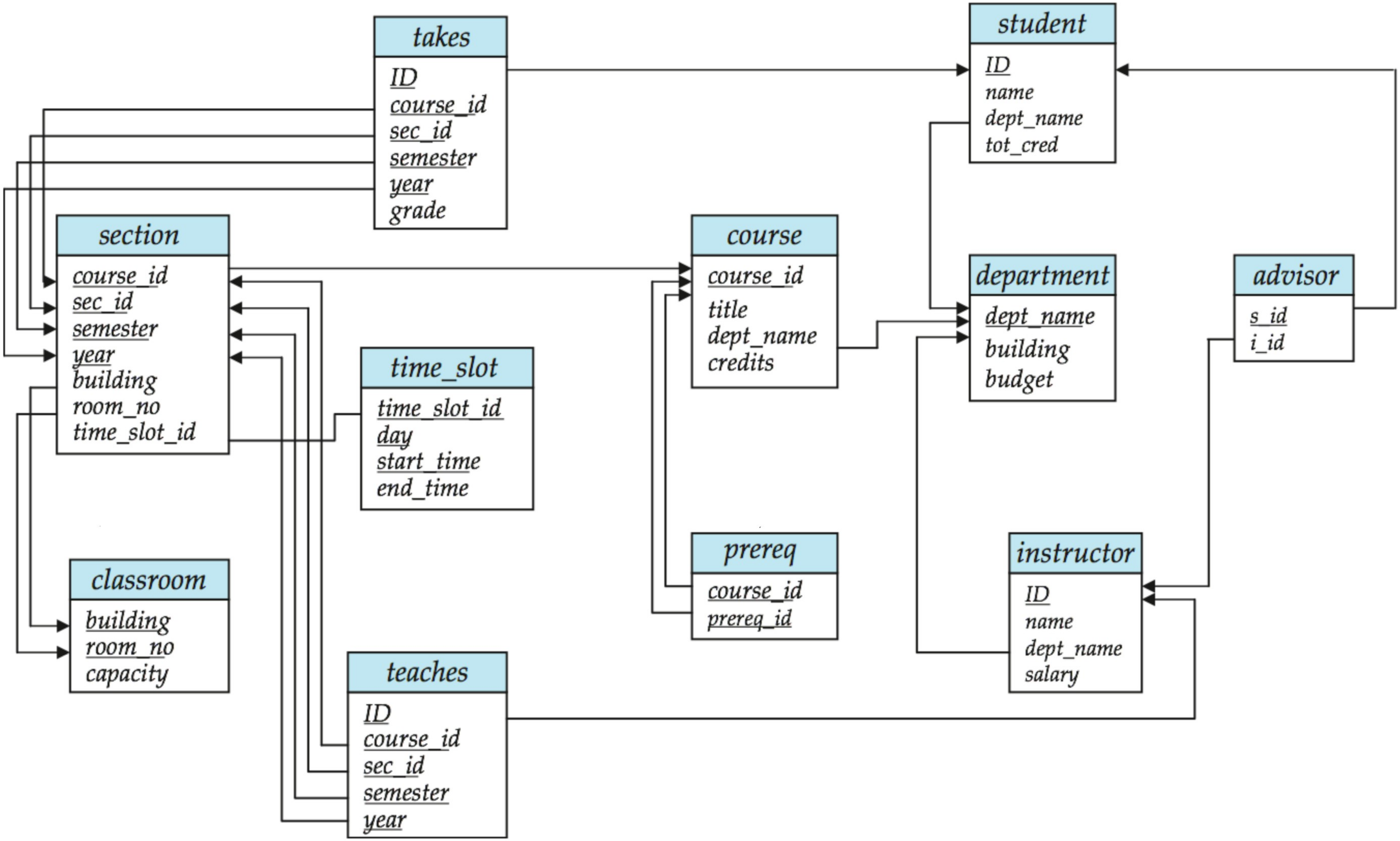
    univ (*instructor -ID, name, dept_name, salary, student_Id*, ..)

    results in
    - repetition of information (e.g., two students have the same instructor)
    - the need for null values (e.g., represent an student with no advisor)
- Normalization theory (Chapter 8) deals with how to design "good" relational schemas

Keys

- Let $K \subseteq R$
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a **candidate key** if *K* is minimal
  Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**.
  - which one?
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
  - **Referenced** relation

Schema Diagram for University Database



Relational Query Languages

- A major strength of the relational model is that it supports simple and powerful *querying* of data.
- Often *declarative* instead of *procedural (imperative)*
- Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.
  - Precise semantics for relational queries.
  - Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

Relational Query Languages

- Procedural vs.non-procedural, or declarative
- "Pure" languages:
  - Relational algebra
    - Procedural
    - Input: two relations; Output: a new relation
  - Tuple relational calculus
    - Non-procedural:
    - E.g., $\{t \mid t \in instructor \wedge t[age] > 40\}$
  - Domain relational calculus
    - Non-procedural:
    - E.g., $\{< i > \mid \exists n, a(< i, n, a > \in instructor \wedge a > 40)\}$
- Relational operators
  - Applied on a single or a pair of relations
  - Result: a single relation

Selection of tuples

- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- Select tuples with A=B and D > 5
  - $\sigma_{\text{A=B and D > 5}}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Selection of Columns (Attributes)

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- Select A and C
  - Projection
  - $\Pi_{A,\,C}\,(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Joining two relations – Cartesian Product

- Relations *r*, *s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Union of two relations

- Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

Set difference of two relations

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- *r* – *s*:

A	B
α	1
β	1

Set Intersection of two relations

- Relation *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Joining two relations – Natural Join

Let r and s be relations on schemas R and S respectively.

Then, the "natural join" of relations R and S is a relation on schema $R \cup S$ obtained as follows:

- Consider each pair of tuples t_r from r and t_s from s.
- If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where
  - t has the same value as t_r on r
  - t has the same value as t_s on s

Natural Join Example

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- Natural Join
  - r ⋈ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Figure in-2.1

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary>=85000}}$ (*instructor*) Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,\ salary}$ (*instructor*) Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
⋈ (Natural Join)	*instructor* ⋈ *department* Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
× (Cartesian Product)	*instructor* × *department* Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
∪ (Union)	Π_{name}(*instructor*) ∪ Π_{name}(*student*) Output the union of tuples from the two input relations.

Examples

Course

course_id	*title*	*dept_name*	*credits*
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4
BIO-399	Computational Biology	Biology	3
CS-101	Intro. to Computer Science	Comp. Sci.	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3
CS-319	Image Processing	Comp. Sci.	3
CS-347	Database System Concepts	Comp. Sci.	3
EE-181	Intro. to Digital Systems	Elec. Eng.	3
FIN-201	Investment Banking	Finance	3
HIS-351	World History	History	3
MU-199	Music Video Production	Music	3
PHY-101	Physical Principles	Physics	4

Dept

dept_name	*building*	*budget*
Biology	Watson	90000
Comp. Sci.	Taylor	100000
Elec. Eng.	Taylor	85000
Finance	Painter	120000
History	Painter	50000
Music	Packard	80000
Physics	Watson	70000

Instructor

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Q1: Find courses in CS department

Q2: Find just id's and titles of courses

Q3: Find courses with > 3 credits

Q4: Find buildings of instructors, and
list instructor name, dept name and building.

Q5: ....

Next

- The Entity- Relationship (ER) model