ECE30030/ITP30010 Database Systems

Normalization

Reading: Chapter 7

Charmgil Hong

charmgil@handong.edu

Spring, 2025

Handong Global University

Announcements

- HW#4 is pre-released
  - Due: Tuesday, May 8, 2025
  - Make sure to check out the problems before the midterm exam

- Reminder: HW#3 is due this Thursday, April 24
  - For Problem 4 (k)-(l), treat *views* as pseudo-tables:
    - (k) Write a query that uses ‘custsomer_list’ as a table.
    - (l) Write an alternative query that does the same as (k) while not using ‘custsomer_list’.

(k) (2 pt.) Using the *'customer_list'* view, list all names of people whose address is in the city of 'London'.
Answer to the question:

Query to find the answer:

(l) (3 pt.) Write a query *that uses only tables (does not use any views)* and returns the same information as in the previous problem (Problem (k)).
Answer to the question:

Query to find the answer:

Announcements

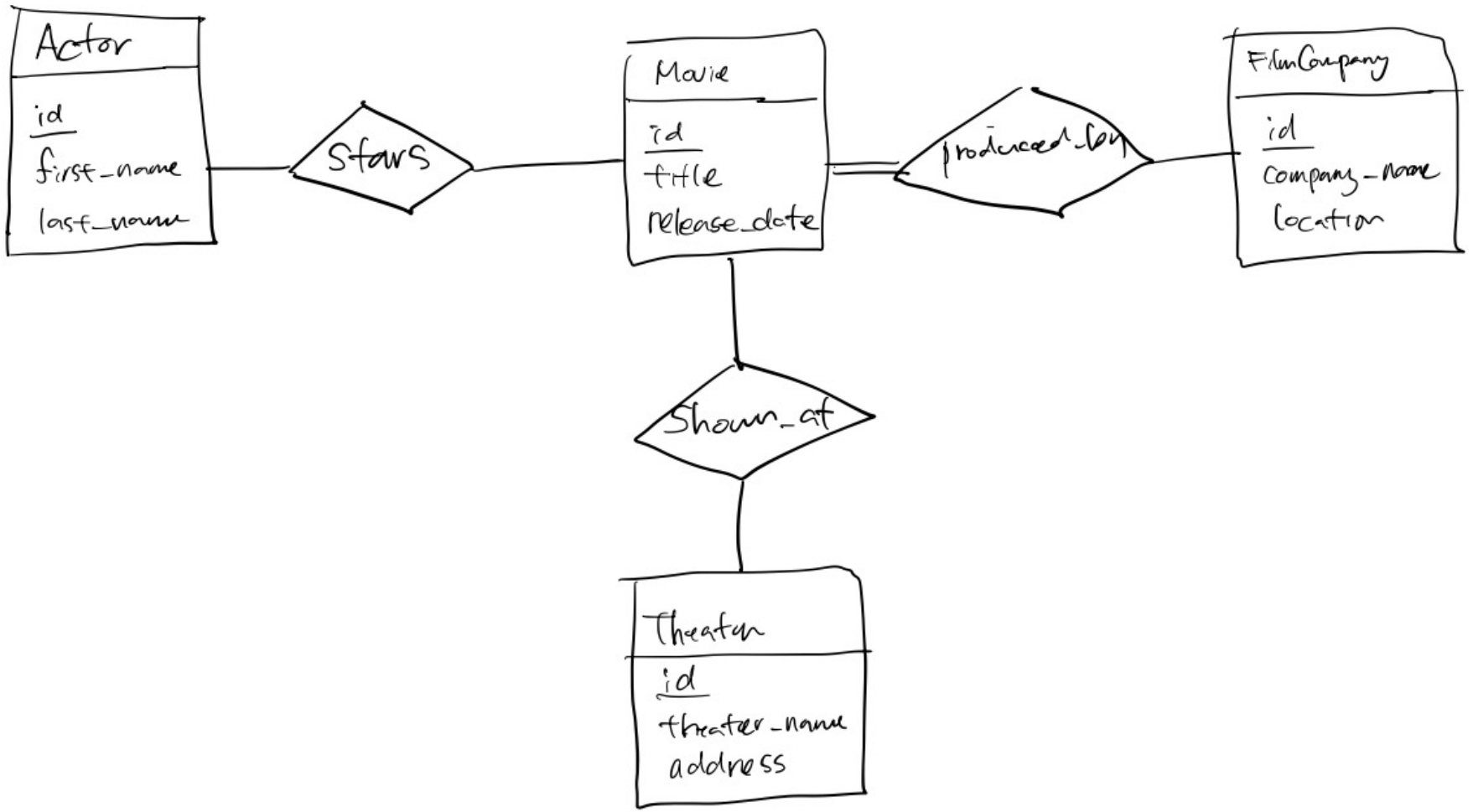
- Midterm is scheduled on Thursday, May 1 (Week #9)
  - Closed-book
  - Covered topics (up-to *DB09ab - Advanced SQL*)
    - Data models
    - Relational data model
    - Relational algebra
    - Structured query language
      - Data manipulation language (including JOIN)
      - Data definition langauge
    - Database design
      - Entity-relationship model
      - Normalization

Announcements

- No offline meeting on May 5 (National holiday)
  - Review on the midterm exam is on Thursday, May 8 (Week #10)

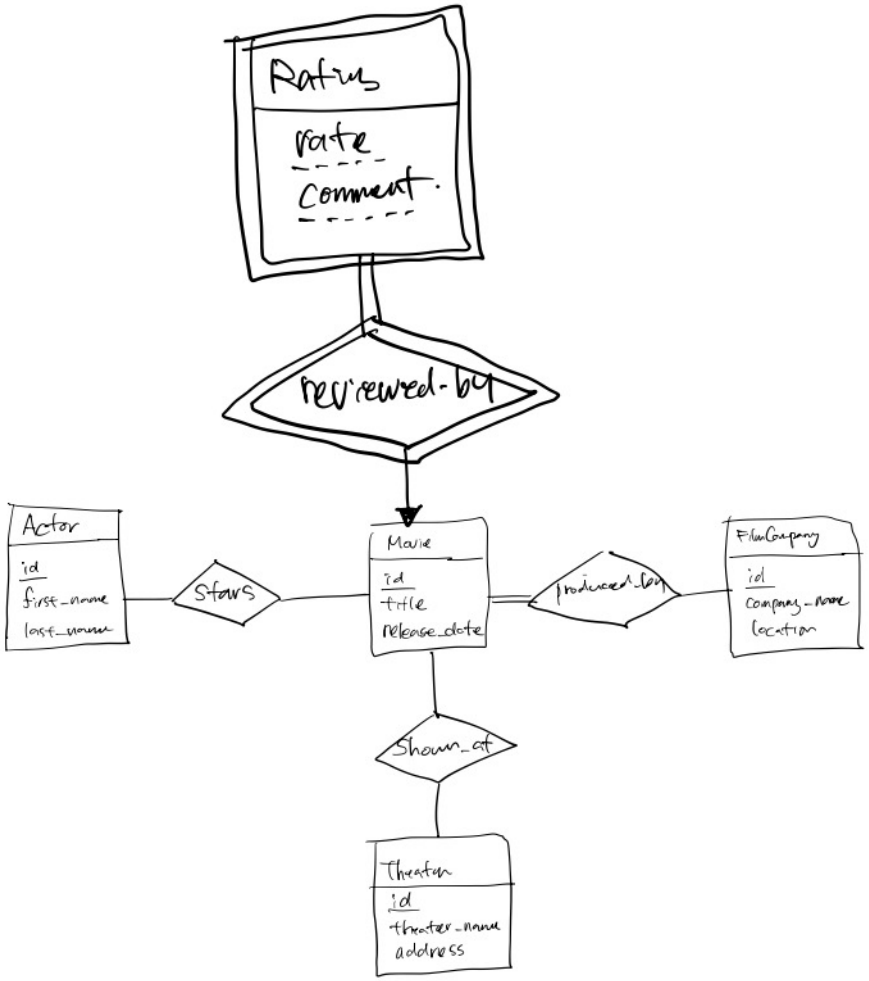
ER-Modal Exercise (Handout)

(a) The *umdb* database



ER-Modal Exercise (Handout)

(b) Adding the *Rating* entity set (and *reviewed_by* relationship set)



ER-Modal Exercise (Handout)

(c) Convert to database schemas (in the set notation)

- Actor(id, first_name, last_name)
- Movie(id, title, release_date)
- Theater(id, theater_name, address)
- FileCompany(id, company_name, location)
- Stars(actor_id, movie_id)
- Shown_at(theater_id, movie_id)
- Produced_by(movie_id, filmcompany_id)
- Rating(movie_id, rate, comment)

Agenda

- Normal forms

When Data is Jumbled…

- Key issues
  - Repetition of data → increases the size of database
  - *Data consistency issues*
    - Insertion anomaly: Inserting redundant data for every new record
    - Deletion anomaly: Loss of related data, when some data is deleted
    - Update anomaly: When updating certain information, every single record must be updated

ID	name	salary	dept_name	building	budget
76766	Crick	72000.00	Biology	Watson	90000.00
10101	Srinivasan	65000.00	Comp. Sci.	Taylor	100000.00
45565	Katz	75000.00	Comp. Sci.	Taylor	100000.00
83821	Brandt	92000.00	Comp. Sci.	Taylor	100000.00
98345	Kim	80000.00	Elec. Eng.	Taylor	85000.00
12121	Wu	90000.00	Finance	Painter	120000.00
76543	Singh	80000.00	Finance	Painter	120000.00
32343	El Said	60000.00	History	Painter	50000.00
58583	Califieri	62000.00	History	Painter	50000.00
15151	Mozart	40000.00	Music	Packard	80000.00

Solution: Decomposition!

- How to avoid the repetition-of-information problem?

ID	name	salary	dept_name	building	budget
76766	Crick	72000.00	Biology	Watson	90000.00
10101	Srinivasan	65000.00	Comp. Sci.	Taylor	100000.00
45565	Katz	75000.00	Comp. Sci.	Taylor	100000.00
83821	Brandt	92000.00	Comp. Sci.	Taylor	100000.00
98345	Kim	80000.00	Elec. Eng.	Taylor	85000.00
12121	Wu	90000.00	Finance	Painter	120000.00
76543	Singh	80000.00	Finance	Painter	120000.00
32343	El Said	60000.00	History	Painter	50000.00
58583	Califieri	62000.00	History	Painter	50000.00
15151	Mozart	40000.00	Music	Packard	80000.00
22222	Einstein	95000.00	Physics	Watson	70000.00
33456	Gold	87000.00	Physics	Watson	70000.00

  - A: Decompose it into two schemas (as they were)
    - Normalization = decomposition of relational schemas
    - Key idea: split relational schemas such that only directly related data composes a relation

Normal Forms

- Normalization process
  - Normalization is a database design technique, which is used to design a relational database table up to higher normal form
    - Procedurally separates logically independent (but related) data entities into multiple relations
    - Maintains the connections using keys
  - Progressive process
    - A higher level of database normalization cannot be achieved unless the previous levels have been satisfied
      - UNF: Unnormalized form
      - 1NF: First normal form
      - 2NF: Second normal form
      - 3NF: Third normal form
      - BCNF: Boyce-Codd normal form (3.5NF)
      - 4NF: Fourth normal form
      - …

First Normal Form (1NF)

- Requirements
  - A relation should consist of atomic values
    - Atomic value: a value that cannot be divided (≃primitive data types in JAVA)
      - Atomic - INT, FLOAT, DOUBLE, DECIMAL (NUMERIC), CHAR, VARCHAR, BLOB, TEXT
      - NOGO - Structure, List (array)
  - Attributes should have unique identifiers

- Step 1 of the normalization process
  - *"If the tables in your DB does not follow 1NF, stop using database"*

First Normal Form (1NF)

- 1NF checklist
  1. Each column should contain an atomic value
     - Entries like (x, y) violate this rule
  2. Each column should contain values that are in the same data domain
     - Do not mix different types of values in a column
  3. Each column should have a unique name
     - Duplicate names lead to confusion while accessing data
  4. The order in which data is stored does not matter
     - Using SQL, one can easily fetch data in any order
  5. There are no duplicated rows in the table
     - Primary key (PK) ensures:
       - Attributes that are part of PK are unique
       - Attributes that are part of PK are not null

First Normal Form (1NF)

- Example

student_id	***name***	***course***
21800999	James Inexistente	Algorithm, OS
21800998	Mike Inexistente	Java
21800997	Matt Inexistente	Algorithm, DB

First Normal Form (1NF)

- Example

student_id	***name***
21800999	James Inexistente
21800998	Mike Inexistente
21800997	Matt Inexistente

student_id	***course***
21800999	Algorithm
21800999	OS
21800998	Java
21800997	Algorithm
21800997	DB

First Normal Form (1NF)

- Example

student_id	***name***
21800999	James Inexistente
21800998	Mike Inexistente
21800997	Matt Inexistente

student_id	***course***
21800999	Algorithm
21800999	OS
21800998	Java
21800997	Algorithm
21800997	DB

Second Normal Form (2NF)

- Requirements
  - A relation should be in 1NF (normal forms should be applied in order)
  - A relation should NOT have a non-PK that is functionally dependent on any subset of any candidate key = NO PARTIAL DEPENDENCIES!
    - Any attributes other than PK should be dependent on PK
    - It should not have partial dependencies
    - PK: Primary Key
      - An attribute or a set of attributes that uniquely identifies each tuple in a relation
      - A PK can fetch data from any specific data in a relation
      - *E.g.*, get the department name of *student_ID* = 21800999

Partial Dependencies

- For relation $R = \underbrace{a_1\ a_2\ a_3}\ a_4$

Primary key (Composite)

  - a_1 is a part of the primary key
    - Partial dependency: a_4 depends on a_1, and does not depend on $a_1a_2a_3$
      - Key: $a_1\ a_2\ a_3$
      - Dependency: $a_1 \rightarrow a_4$

Eliminating Partial Dependencies

- Example: *score*(*score_id*, *student_id*, *subject_id*, *score*, *instructor*)
  - *subject_id* → *instructor* : *partial dependency*

score_id	***student_id***	***subject_id***	***score***	***instructor***
1	10	1	82	James Packer
3	11	1	95	James Packer
2	10	2	77	Cole Miller
4	11	2	71	Cole Miller
5	11	4	96	Adam Lee

Eliminating Partial Dependencies

- Example: *score_a*(*score_id*, *student_id*, *subject_id*, *score*)
  score_b(*subject_id*, *instructor*)

score_id	*student_id*	*subject_id*	*score*
1	10	1	82
3	11	1	95
2	10	2	77
4	11	2	71
5	11	4	96

subject_id	*instructor*
1	James Packer
2	Cole Miller
4	Adam Lee

Third Normal Form (3NF)

- Requirements
  - A relation should be in 2NF
  - A relation should NOT have transitive dependencies
    - Transitive dependency: A non-PK attribute depends on another non-PK attribute or a set of non-PK attributes

Transitive Dependencies

- For relation $R = a_1\ a_2\ a_3\ a_4$

Candidate key = a1, a2, {a1,a2}

  - a_1 is the primary key in R
    - a_3 depends on a_1 ($a_1 \rightarrow a_3$) -- OK
    - a_4 depends on a_3 ($a_1 \rightarrow a_3 \rightarrow a_4$) … non-PK → non-PK

Eliminating Transitive Dependencies

- Example

BookNo	Patron	Address	Due
B1	J. Fisher	101 Main Street	3/2/15
B2	L. Perez	202 Market Street	2/28/15

  - PK: BookNo
  - Patron → Address

Eliminating Transitive Dependencies

- Example

BookNo	Patron	Due
B1	J. Fisher	3/2/15
B2	L. Perez	2/28/15

Patron	Address
J. Fisher	101 Main Street
L. Perez	202 Market Street

Boyce-Codd Normal Form (BCNF) = 3.5NF

- Requirements
  - A relation should be in 3NF
  - For A → B, if A is non-PK, then it is NOT in BCNF
    - For any dependency A → B, A should be a super key
    - There should be no overlapping candidate keys

Boyce-Codd Normal Form (BCNF)

- Example: *takes2*(*student_id*, *subject*, *instructor*)

student_id	*subject*	*instructor*
21800999	C++	Dr. Cpp
21800999	Java	Dr. Java
21800998	C++	Dr. C
21800997	Python	Dr. Python
21800996	C++	Dr. Cpp

- (student, subject) → instructor
- Instructor → subject
  - A non-PK identifies a member of PK: Not in BCNF

Boyce-Codd Normal Form (BCNF)

- Example: *takes2a*(*student_id*, *section_id*),
  takes2b(*section_id*, *subject*, *instructor*)

student_id	*section_id*
21800999	101
21800999	103
21800998	102
21800997	104
21800996	101

section_id	*subject*	*instructor*
101	C++	Dr. Cpp
103	Java	Dr. Java
102	C++	Dr. C
104	Python	Dr. Python

Boyce-Codd Normal Form (BCNF)

- Example: *takes2a*(*student_id*, *section_id*),
  takes2b(*section_id*, *subject*, *instructor*)

student_id	*section_id*
21800999	101
21800999	103
21800998	102
21800997	104
21800996	101

section_id	*subject*	*instructor*
101	C++	Dr. Cpp
103	Java	Dr. Java
102	C++	Dr. C
104	Python	Dr. Python

Fourth Normal Form (4NF)

- Requirements
  - A relation should be in BCNF
  - A relation should NOT have multi-valued dependency
    - Multi-valued dependency occurs due to a bad DB schema

    - Multi-valued dependency occurs when a relation has more than 3 attributes
    - For a relation with attributes A, B, C
      - having dependency, A → B, and
      - B and C are independent from each other
        ➔ Then, the relation may have a multi-valued dependency

Multi-valued Dependencies

- Example
  - *student_id* → *course*
  - *student_id* → *activity*
  - *course* ⊥ *activity* (independent)

student_id	***course***	***activity***
21800999	Statistics	Soccer
21800999	Linear algebra	Basketball
21800999	Statistics	Basketball
21800999	Linear algebra	Soccer

Multi-valued Dependencies

- Example

student_id	*course*
21800999	Statistics
21800999	Linear algebra
21800998	Physics
21800998	Programming 101

student_id	*activity*
21800999	Soccer
21800999	Basketball
21800998	Pool
21800997	Soccer

Summary: Normal Forms

- Theories – Normal forms

Theory	Key Idea	Normal Form
Functional dependency	(PK → non-PK)	2NF
Partial dependency	Part of PK → non-PK	2NF
Transitive dependency	Non-PK → non-PK	3NF
-	Non-PK → PK	BCNF
Multi-valued dependency		4NF
Join dependency		5NF

Overall DB Design Process

- Let us assume schema *R* is given:

  (E-R Model)

  - *R* could have been generated when converting E-R diagram to a set of tables

  (Normalization)

  - *R* could have been a single relation containing *all* attributes that are of interest (called universal relation)
  - Normalization breaks *R* into smaller relations

  (Mixed)

  - *R* could have been the result of some ad-hoc design of relations, which we then test/convert to normal form

E-R Model and Normalization

- When an E-R diagram is carefully designed, identifying all entities correctly, the tables generated from the E-R diagram should not need further normalization

- However, in a real (imperfect) design, there can be functional dependencies from non-key attributes of an entity to other attributes of the entity
  - *E.g.*, an *employee* entity with
    - attributes
      department_name and *building*
    - functional dependency
      department_name $\rightarrow$ *building*
    - Good design would have made *department* an entity

- Functional dependencies from non-key attributes of a relationship set possible, but rare --- most relationships are binary

Denormalization for Performance

- We may want to use non-normalized schema for performance
  - For example, displaying *prereqs* along with *course_id,* and *title* requires join of *course* with *prereq*

- Alternative 1: Use denormalized relation containing attributes of *course* as well as *prereq* with all above attributes
  - faster lookup
  - extra space and extra execution time for updates
  - extra coding work for programmer and possibility of error in extra code

- Alternative 2: Use a materialized view defined a *course* ⋈ *prereq*
  - Benefits and drawbacks same as above, except no extra coding work for programmer and avoids possible errors

Remaining Issues

- Some aspects of database design are not caught by normalization

- Example (to be avoided)
  - *earnings_2004, earnings_2005, earnings_2006*, etc., all on the schema (*company_id, earnings*)
    - Above are well normalized (in BCNF), but make querying across years difficult and needs new table each year
  - *company_year* (*company_id, earnings_2004, earnings_2005, earnings_2006*)
    - Above are well normalized (in BCNF), but makes querying across years difficult and requires new attribute each year

  - This is an example of a crosstab, where values for one attribute become column names

  → Better schema: *earnings* (*company_id, year, amount*)

ECE30030/ITP30010 Database Systems

Advanced SQL

Reading: Chapters 4-5

Charmgil Hong

charmgil@handong.edu

Spring, 2024

Handong Global University

Agenda

- Join

Join Operations

- Join operations take two relations and return another relation
  - A join is a Cartesian product that requires tuples in the two relations match
    - It also specifies the attributes that are present in the result of the join (project)
  - Typically used as subquery expressions in the **FROM** clause

  - Join types
    - **INNER JOIN**
    - **OUTER JOIN**

  - Join conditions
    - **NATURAL**
    - **ON** <predicate>
    - **USING (**A_1, A_2, …, A_n**)**

Running Example

- Relations: student, takes

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58
76653	Aoi	Elec. Eng.	60
98765	Bourikas	Elec. Eng.	98
98988	Tanaka	Biology	120

ID	course_id	sec_id	semester	year	grade
00128	CS-101	1	Fall	2017	A
00128	CS-347	1	Fall	2017	A-
12345	CS-101	1	Fall	2017	C
12345	CS-190	2	Spring	2017	A
12345	CS-315	1	Spring	2018	A
12345	CS-347	1	Fall	2017	A
19991	HIS-351	1	Spring	2018	B
23121	FIN-201	1	Spring	2018	C+
44553	PHY-101	1	Fall	2017	B-
45678	CS-101	1	Fall	2017	F
45678	CS-101	1	Spring	2018	B+
45678	CS-319	1	Spring	2018	B
54321	CS-101	1	Fall	2017	A-
54321	CS-190	2	Spring	2017	B+
55739	MU-199	1	Spring	2018	A-
76543	CS-101	1	Fall	2017	A
76543	CS-319	2	Spring	2018	A
76653	EE-181	1	Spring	2017	C
98765	CS-101	1	Fall	2017	C-
98765	CS-315	1	Spring	2018	B
98988	BIO-101	1	Summer	2017	A
98988	BIO-301	1	Summer	2018	*<null>*

Running Example

- Relations: course, instructor

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

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000.00
12121	Wu	Finance	90000.00
15151	Mozart	Music	40000.00
22222	Einstein	Physics	95000.00
32343	El Said	History	60000.00
33456	Gold	Physics	87000.00
45565	Katz	Comp. Sci.	75000.00
58583	Califieri	History	62000.00
76543	Singh	Finance	80000.00
76766	Crick	Biology	72000.00
83821	Brandt	Comp. Sci.	92000.00
98345	Kim	Elec. Eng.	80000.00

Natural Join

- Natural join matches tuples with the same values for all common attributes, and retains only one copy of each common column
  - *E.g.*, List the names of students along with the course ID of the courses that they took
    - **SELECT** *name*, *course_id*
      FROM *student, takes*
      WHERE *student.ID = takes.ID*;
  - Same query in SQL with natural join:
    - **SELECT** *name*, *course_id*
      FROM *student* **NATURAL JOIN** *takes*;

name	course_id
Zhang	CS-101
Zhang	CS-347
Shankar	CS-101
Shankar	CS-190
Shankar	CS-315
Shankar	CS-347
Brandt	HIS-351
Chavez	FIN-201
Peltier	PHY-101
Levy	CS-101
Levy	CS-101
Levy	CS-319
Williams	CS-101
Williams	CS-190
Sanchez	MU-199
Brown	CS-101
Brown	CS-319
Aoi	EE-181
Bourikas	CS-101
Bourikas	CS-315
Tanaka	BIO-101
Tanaka	BIO-301

Natural Join

- The **FROM** clause can have multiple relations combined using natural join:
  - **SELECT** $A_1, A_2, \ldots A_n$
    FROM r_1 **NATURAL JOIN** r_2 **NATURAL JOIN** … **NATURAL JOIN** r_n
    WHERE P ;

Caveat

- *E.g.*, **(Incorrect)**
  SELECT *dept_name, course_id, name, title, credits*
  FROM *student* **NATURAL JOIN** *takes* **NATURAL JOIN** *course*;

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

Caveat

- Beware of unrelated attributes with same name getting equated incorrectly
  - *E.g.*, List the names of students along with the titles of courses that they have taken
    - Correct

      SELECT *name*, *title*
      FROM *student* **NATURAL JOIN** *takes*, *course*
      WHERE *takes.course_id* = *course.course_id*;

    - Incorrect

      SELECT *name*, *title*
      FROM *student* **NATURAL JOIN** *takes* **NATURAL JOIN** *course*;

      - This query omits all (student name, course title) pairs where the student takes a course in a department other than the student's own department

Join with USING Clause

- To avoid the danger of equating attributes erroneously, use the **USING** construct
  - USING: allows us to specify exactly which columns should be equated
  - *E.g.*, **SELECT** *name*, *title*
    FROM (*student* **NATURAL JOIN** *takes*) **JOIN** *course* **USING (***course_id***)**

name	title
Tanaka	Intro. to Biology
Tanaka	Genetics
Zhang	Intro. to Computer Science
Shankar	Intro. to Computer Science
Levy	Intro. to Computer Science
Williams	Intro. to Computer Science
Brown	Intro. to Computer Science
Bourikas	Intro. to Computer Science
Levy	Intro. to Computer Science
Shankar	Game Design
Williams	Game Design
Shankar	Robotics
Bourikas	Robotics
Levy	Image Processing
Brown	Image Processing
Zhang	Database System Concepts
Shankar	Database System Concepts
Aoi	Intro. to Digital Systems
Chavez	Investment Banking
Brandt	World History
Sanchez	Music Video Production
Peltier	Physical Principles

JOIN … ON

- The **ON** condition allows a general predicate over the relations being joined
  - Written like a **WHERE** clause predicate
  - *E.g.*, **SELECT** *
    FROM *student* **JOIN** *takes* **ON** *student.ID = takes.ID*
    - The **ON** condition specifies that a tuple from *student* matches a tuple from *takes* if their *ID* values are equal
    - Equivalent to:
      SELECT *name*, *course_id*
      FROM *student, takes*
      WHERE *student.ID = takes.ID*;

Inner Join

- Inner join: Does not preserve nonmatched tuples
  - Tables are joined based on common columns mentioned in the **ON** or **USING** clause
  - One can specify the condition with an **ON** or **USING** construct

- *C.f.*, Natural join: assumes the join condition to be where same-named columns in both tables match
  - One cannot use **ON** or **USING**
  - In the result of a natural join, repeated columns are avoided

Natural Join

- Natural join: Some tuples in either or both relations being joined may be lost
  - *E.g.,* **SELECT** *
    FROM *course* **NATURAL JOIN** *prereq*;

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101

Examples

- Tables

ROLL_NO	NAME
1	HARSH
2	PRATIK
3	RIYANKA
4	DEEP
5	SAPTARHI
6	DHANRAJ
7	ROHIT
8	NIRAJ

Student

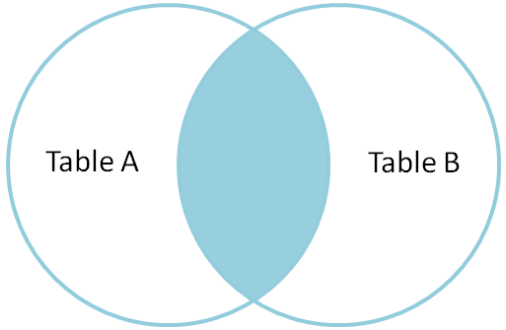
COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

StudentCourse

Examples

- Inner join
  - **SELECT** *StudentCourse.COURSE_ID, Student.NAME*
    FROM *Student*
    INNER JOIN *StudentCourse*
    ON *Student.ROLL_NO = StudentCourse.ROLL_NO;*

COURSE_ID	NAME
1	HARSH
2	PRATIK
2	RIYANKA
3	DEEP
1	SAPTARHI



Examples

- Left join
  - **SELECT** *Student.NAME, StudentCourse.COURSE_ID*
    FROM *Student*
    LEFT JOIN *StudentCourse*
    ON *StudentCourse.ROLL_NO = Student.ROLL_NO;*

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

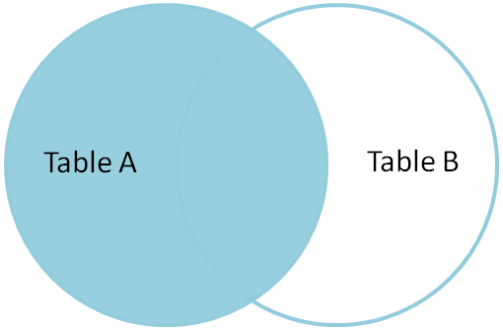
- Right join
  - **SELECT** *Student.NAME, StudentCourse.COURSE_ID*
    FROM *Student*
    RIGHT JOIN *StudentCourse*
    ON *StudentCourse.ROLL_NO = Student.ROLL_NO;*

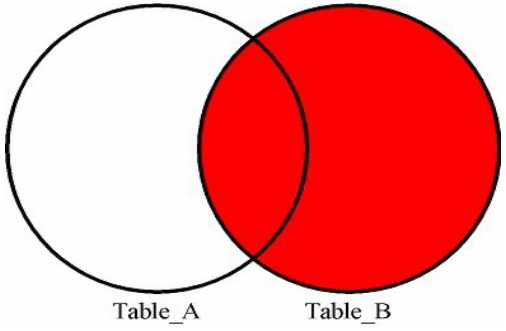
NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

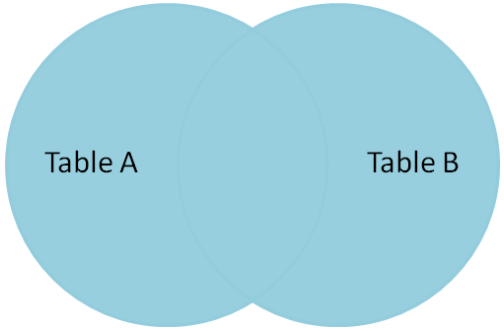
Examples

- Full join
  - **SELECT** *Student.NAME, StudentCourse.COURSE_ID*
    FROM *Student*
    FULL JOIN *StudentCourse*
    ON *StudentCourse.ROLL_NO = Student.ROLL_NO;*

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*
NULL	9
NULL	10
NULL	11







Join Types