
Lesson 4.1: Relational DBs



UMD DATA605 - Big Data Systems

Lesson 4.1: Relational DBs

Instructor: Dr. GP Saggese, gsaggese@umd.edu



1 / 13

Relational Model: Overview

- Introduced by Ted Codd (late 60's, early 70's)
- **First prototypes**
 - Ingres Project at Berkeley (1970-1985)
 - Ingres (INteractive Graphics REtrieval System)
 - → PostgreSQL (=Post Ingres)
 - IBM System R (1970s) → Oracle, IBM DB2
- **Contributions from relational data model**
 - Formal semantics for data operations
 - Data independence: separation of logical and physical data models
 - Declarative query languages (e.g., SQL)
 - Query optimization
- **Key to commercial success**



2 / 13

- **Relational Model: Overview**
 - The relational model was introduced by Ted Codd in the late 1960s and early 1970s. This model revolutionized how databases were structured and accessed, laying the groundwork for modern database systems.
- **First prototypes**
 - The Ingres Project at Berkeley, which ran from 1970 to 1985, was one of the first implementations of the relational model. Ingres stands for INteractive Graphics REtrieval System and was a pioneering project that eventually led to the development of PostgreSQL, a widely used database system today.
 - IBM System R was another early implementation in the 1970s. It was a research project that significantly influenced the development of commercial database systems like Oracle and IBM DB2.
- **Contributions from relational data model**
 - The relational model provided formal semantics for data operations, which means it defined a clear and mathematical way to handle data.
 - It introduced the concept of data independence, allowing the separation of logical data models (how data is organized) from physical data models (how data is stored), making databases more flexible and easier to manage.
 - Declarative query languages, such as SQL, emerged from this model, enabling users to specify what data they want without detailing how to retrieve it.
 - Query optimization became possible, allowing databases to efficiently execute queries by determining the best way to access and process data.
- **Key to commercial success**
 - The relational model's ability to provide a structured and efficient way to manage data

was crucial to its widespread adoption in commercial applications. Its principles continue to underpin many modern database systems.

3 / 13: Relational Model: Key Definitions

Relational Model: Key Definitions

- Relational DB is a collection of **tables / relations**
 - Unique name and schema for each table
 - E.g., `instructor` and `course` relations
- **Row / tuple / record**: Represents a relationship among values
- **Element**: Corresponds to a **column / field / attribute**
 - Atomic elements (e.g., phone number as a single object)
 - NULL for unknown or non-existent values
- **Schema of a relation**
 - List of attributes and their domains
 - Like type definition in programming languages
 - E.g., domain of `salary` is integers ≥ 0
- **Instance of relation**
 - Specific instantiation with actual values
 - Changes over time

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	Califleri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

`instructor` relation

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

`course` relation



3 / 13

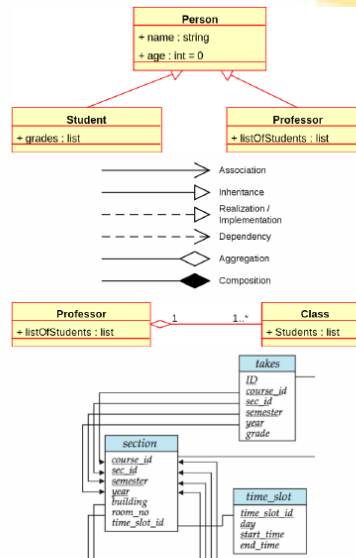
- **Relational DB is a collection of tables / relations**: In a relational database, data is organized into tables, also known as relations. Each table has a unique name and a defined structure, called a schema. For example, you might have tables named `instructor` and `course`, each with its own set of columns and data types.
- **Row / tuple / record**: Each row in a table represents a single, complete set of related data. Think of it as a single entry in a table that connects different pieces of information, like a row in the `instructor` table that includes an instructor's ID, name, and department.
- **Element**: This refers to the individual data points within a row, corresponding to columns or fields. Each element is atomic, meaning it holds a single piece of data, like a phone number. If data is missing or unknown, it is represented by NULL.
- **Schema of a relation**: The schema defines the structure of a table, listing all the attributes (columns) and their data types or domains. It's similar to a type definition in programming, ensuring that data fits expected formats, like ensuring a `salary` is always a non-negative integer.
- **Instance of relation**: This is a snapshot of the table at a particular moment, filled with actual data. As data is added, removed, or updated, the instance of the relation changes over time, reflecting the current state of the database.

The images on the right visually represent the `instructor` and `course` tables, showing how data is organized in a relational database.

4 / 13: UML Class Diagram

UML Class Diagram

- **UML class diagram**
 - UML = Unified Modeling Language
 - Used in OOP and DB design
- **In OOP design**
 - Diagram showing classes, attributes, methods, and relationships
- **In DB design**
 - Each box is a table / relation
 - Columns / fields / attributes are listed inside the box
 - Primary keys are underlined
 - Foreign key constraints are represented by arrows



4 / 13

- **UML class diagram**
 - UML stands for *Unified Modeling Language*, which is a standardized way to visualize the design of a system.
 - It is commonly used in *Object-Oriented Programming (OOP)* and database (DB) design to help developers and designers understand the structure and relationships within a system.
- **In OOP design**
 - A UML class diagram is a visual representation that shows the classes in a system, along with their attributes (properties) and methods (functions).
 - It also illustrates the relationships between different classes, such as inheritance, association, and dependency, which helps in understanding how different parts of the system interact with each other.
- **In DB design**
 - In the context of database design, each box in a UML class diagram represents a table or a relation.
 - Inside each box, the columns or fields of the table are listed, which are also known as attributes.
 - Primary keys, which uniquely identify each record in a table, are underlined to distinguish them from other attributes.
 - Foreign key constraints, which establish relationships between tables, are depicted by arrows pointing from one table to another, indicating how tables are linked.

This slide provides a foundational understanding of how UML class diagrams are used in both software and database design, emphasizing their role in visualizing and organizing complex systems.

5 / 13: Example: University DB

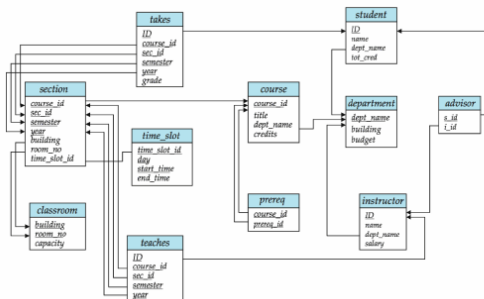
Example: University DB

- **UML diagram of a DB and schemas representing a University**

- Each box is a table / relation
- Column / fields / attributes are listed inside the box
- Primary keys are underlined fields
- Foreign key constraints are arrows between boxes

- **Analysis of the diagram**

- ER model
- Entities
 - student
 - department
 - ...
- Relationships
 - takes
 - teaches
 - ...



5 / 13

- **UML diagram of a DB and schemas representing a University**

- The diagram is a visual representation of a database using UML (Unified Modeling Language). It helps us understand how data is organized within a university's database.
- **Each box is a table / relation:** In the diagram, each box represents a table in the database. A table is a collection of related data entries, similar to a spreadsheet.
- **Column / fields / attributes are listed inside the box:** Inside each box, you'll see a list of fields or attributes. These are the individual pieces of data stored in the table, like a student's name or ID.
- **Primary keys are underlined fields:** A primary key is a unique identifier for each record in a table. It's underlined in the diagram to show its importance.
- **Foreign key constraints are arrows between boxes:** Arrows indicate relationships between tables. A foreign key in one table points to a primary key in another, linking the data.

- **Analysis of the diagram**

- **ER model:** The diagram is based on an Entity-Relationship (ER) model, which is a way to visually represent data and its relationships.
- **Entities:** These are the main objects or concepts in the database. For example:
 - * **student:** Represents students in the university.
 - * **department:** Represents different departments within the university.
 - * **...:** There are likely more entities, each representing a different aspect of the university.
- **Relationships:** These show how entities are connected. For example:
 - * **takes:** Represents the relationship between students and the courses they enroll in.
 - * **teaches:** Represents the relationship between instructors and the courses they

teach.

* ...: Additional relationships help define how entities interact with each other.

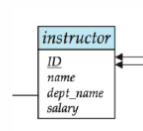
6 / 13: Primary Key

Primary Key

- R is a set of attributes of a relation r
 - E.g., ID, name, dept_name, salary are attributes of instructor
- K is a superkey of R if values for K identify a unique tuple of each relation $r(R)$
 - E.g., (ID) and (ID, name) are superkeys of instructor
 - (name) is not a superkey of instructor
- **Primary key**: minimal set of attributes that uniquely identify each row
 - Typically small and immutable
 - Would SSN be a primary key? Yes and no
- **Primary key constraint**: rows can't have the same primary key

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	Califleri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor relation



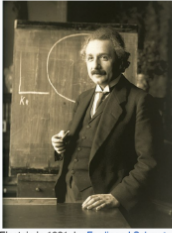
6 / 13

- **R is a set of attributes of a relation r** : In databases, a relation is like a table, and attributes are the columns in that table. For example, in an **instructor** table, the columns might be ID, name, dept_name, and salary. These columns hold specific types of data for each instructor.
- **K is a superkey of R** : A superkey is a set of one or more columns that can uniquely identify any row in the table. For instance, in the **instructor** table, the ID column alone can uniquely identify each instructor, making it a superkey. Even a combination like (ID, name) is a superkey, but it's not minimal. However, just name isn't a superkey because multiple instructors could have the same name.
- **Primary key**: This is the smallest set of columns that can uniquely identify each row in the table. It's important for a primary key to be small and not change over time. For example, a Social Security Number (SSN) could be a primary key because it's unique to each person, but it might not be ideal due to privacy concerns.
- **Primary key constraint**: This rule ensures that no two rows in the table can have the same primary key value. This is crucial for maintaining the uniqueness of each row in the database, preventing duplicate entries.

7 / 13: Question: What Are Primary Keys?

Question: What Are Primary Keys?

- Marital status
 - Married(*person1_ssn*, *person2_ssn*, *date_married*, *date_divorced*)
- Bank account
 - Account(*cust_ssn*, *account_number*, *cust_name*, *balance*, *cust_address*)
- Research assistantship at UMD
 - RA(*student_id*, *project_id*, *supervisor_id*, *appt_time*, *appt_start_date*, *appt_end_date*)
- Information typically found on Wikipedia
 - Person(*Name*, *Born*, *Died*, *Citizenship*, *Education*, ...)
- Info about US President on Wikipedia
 - President(*name*, *start_date*, *end_date*, *vice_president*, *preceded_by*, *succeeded_by*)
- Tour de France: historical rider participation information
 - Rider(*Name*, *Born*, *Team-name*, *Coach*, *Sponsor*, *Year*)

Albert Einstein	
	
Einstein in 1921, by Ferdinand Schmutzer	
Born	14 March 1879 Ulm, Germany
Died	18 April 1955 (aged 76) Princeton, New Jersey, U.S.
Citizenship	Full list [show]
Education	Federal polytechnic school in Zurich (Federal teaching diploma, 1900) University of Zurich (PhD, 1905)
Known for	General relativity Special relativity Photoelectric effect $E=mc^2$ (Mass–energy equivalence) $E=h\nu$ (Planck–Einstein relation) Theory of Brownian motion



7 / 13

- **Question: What Are Primary Keys?**
 - **Marital status**
 - * This example shows a table that might be used to track marital status. The primary key here could be a combination of *person1_ssn* and *person2_ssn*, as these two together uniquely identify a marriage record. The *date_married* and *date_divorced* provide additional context but are not unique identifiers.
 - **Bank account**
 - * In this table, the primary key is likely the *account_number*, as it uniquely identifies each bank account. Other fields like *cust_ssn* and *cust_name* provide additional information but are not unique by themselves.
 - **Research assistantship at UMD**
 - * Here, the primary key could be a combination of *student_id* and *project_id*, as this combination uniquely identifies each research assistantship. The other fields provide details about the appointment.
 - **Information typically found on Wikipedia**
 - * For a general person entry, a primary key might not be explicitly defined, but a unique identifier could be a combination of *Name* and *Born* date, as these together can uniquely identify a person.
 - **Info about US President on Wikipedia**
 - * The primary key could be a combination of *name* and *start_date*, as these together uniquely identify a presidential term. Other fields provide context about the presidency.
 - **Tour de France: historical rider participation information**
 - * In this table, a primary key could be a combination of *Name* and *Year*, as these

together uniquely identify a rider's participation in a specific year. Other fields provide additional details about the rider's participation.

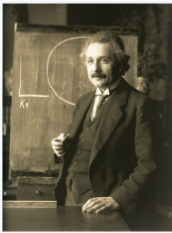
- **Image Context**

- The image likely provides a visual representation of how primary keys are used in databases to uniquely identify records. It might show examples of tables with highlighted primary keys to reinforce the concept.

8 / 13: Answer: What Are Primary Keys?

Answer: What Are Primary Keys?

- Marital status
 - Married(**person1_ssn**, **person2_ssn**, **date_married**, date_divorced)
- Bank account
 - Account(cust_ssn, **account_number**, cust_name, balance, cust_address)
- Research assistantship at UMD
 - RA(**student_id**, **project_id**, supervisor_id, appt_time, **appt_start_date**, appt_end_date)
- Information typically found on Wikipedia
 - Person(**Name**, **Born**, **Died**, **Citizenship**, **Education**, ...)
- Info about US President on Wikipedia
 - President(**name**, **start_date**, end_date, vice_president, preceded_by, succeeded_by)
- Tour de France: historical rider participation information
 - Rider(**Name**, **Born**, **Team-name**, Coach, Sponsor, **Year**)

Albert Einstein	
	
Einstein in 1921, by Ferdinand Schmutzer	
Born	14 March 1879 Ulm, Germany
Died	18 April 1955 (aged 76) Princeton, New Jersey, U.S.
Citizenship	Full list [show]
Education	Federal polytechnic school in Zurich (Federal teaching diploma, 1900) University of Zurich (PhD, 1905)
Known for	General relativity Special relativity Photoelectric effect $E=mc^2$ (Mass–energy equivalence) $E=h\nu$ (Planck–Einstein relation) Theory of Brownian motion



8 / 13

- **Marital Status**

- In a database table for marital status, the primary key is a combination of **person1_ssn**, **person2_ssn**, and **date_married**. This means that each marriage record is uniquely identified by the social security numbers of the two individuals involved and the date they got married. This combination ensures that each marriage is distinct, even if the same individuals marry multiple times.

- **Bank Account**

- For bank accounts, the primary key is the **account_number**. This is a unique identifier for each bank account, ensuring that no two accounts have the same number. It helps in efficiently managing and accessing account details like customer name, balance, and address.

- **Research Assistantship at UMD**

- In the context of research assistantships, the primary keys are **student_id** and **project_id**. This combination uniquely identifies each assistantship position, linking a student to a specific project. Additionally, **appt_start_date** is also a key, which might be used to track the duration of the assistantship.

- **Information Typically Found on Wikipedia**

- For a person’s information on Wikipedia, the primary key is a combination of **Name, Born, Died, Citizenship, Education, ...**. This set of attributes uniquely identifies a person, considering that names alone might not be unique.
- **Info About US President on Wikipedia**
 - The primary keys for US Presidents are **name and start_date**. This combination ensures that each presidential term is uniquely identified, even if a president serves non-consecutive terms.
- **Tour de France: Historical Rider Participation Information**
 - For historical rider participation in the Tour de France, the primary keys are **Name, Born, Team-name, and Year**. This combination uniquely identifies a rider’s participation in a specific year, accounting for changes in teams or multiple participations over different years.

In summary, primary keys are crucial in databases as they uniquely identify each record, ensuring data integrity and efficient data retrieval.

9 / 13: Foreign Key

Foreign Key

- **Foreign key** = primary key of another relation
 - E.g., (ID) from student in takes, advisor
 - takes is the “referencing relation”, has the foreign key
 - student is the “referenced relation”, has the primary key
 - Shown by an arrow from referencing → referenced
- **Foreign key constraint**: for each row, the primary key tuple must exist
 - Aka referential integrity constraint
 - If (student101, DATA605) is in takes, there must be student101 in student
- The key referenced as a foreign key must exist as a primary key



- **Foreign key**: This is a concept in databases where a column (or a set of columns) in one table is used to refer to the primary key of another table.
 - For example, if you have a column (ID) in a table called **takes**, it might be used to refer to the ID in another table called **student**. This means that the ID in **takes** is a foreign key.
 - The table **takes** is known as the “referencing relation” because it contains the foreign key.
 - The table **student** is the “referenced relation” because it contains the primary key that the foreign key points to.

- This relationship is often depicted with an arrow pointing from the referencing table to the referenced table.
- **Foreign key constraint:** This is a rule that ensures data integrity between tables.
 - Also known as the referential integrity constraint, it requires that for every foreign key value in the referencing table, there must be a corresponding primary key value in the referenced table.
 - For instance, if there is an entry (`student101`, `DATA605`) in the `takes` table, there must be a corresponding `student101` entry in the `student` table.
 - This ensures that the foreign key always points to a valid, existing primary key in the referenced table.

10 / 13: Relational Algebra: 1/4

Relational Algebra: 1/4

- **Relation:** set of tuples
- **Relational algebra:** operations on relations producing a new relation
 - Unary: selection, projection, rename
 - Binary: union, set difference, intersection, Cartesian product, join
- **Selection σ :** select tuples satisfying a predicate
 - E.g., select instructor tuples where `dept_name = "Physics"`
- **Projection π :** return tuples with subset of attributes
 - E.g., project instructor tuples with (`name`, `salary`)
- **Set operations:** union, intersection, set difference
 - Must be compatible (same attributes)

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	Califleri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

ID	name	dept_name	salary
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

$\sigma_{dept_name = 'Physics'}(instructor)$

ID	name	salary
10101	Srinivasan	65000
12121	Wu	90000
15151	Mozart	40000
22222	Einstein	95000
32343	El Said	60000
33456	Gold	87000
45565	Katz	75000
58583	Califleri	62000
76543	Singh	80000
76766	Crick	72000
83821	Brandt	92000
98345	Kim	80000

$\Pi_{ID, name, salary}(instructor)$

10 / 13



- **Relation:** In the context of databases, a *relation* is essentially a table. Each table consists of rows, known as *tuples*, and columns, which are the attributes of the data. Think of a relation as a structured way to store data where each row represents a unique data entry.
- **Relational algebra:** This is a set of operations used to manipulate and query data stored in relations. The result of these operations is always a new relation. Relational algebra is foundational for understanding how queries are processed in databases.
 - **Unary operations:** These operations work on a single relation.
 - * *Selection* (σ) filters rows based on a condition.
 - * *Projection* (π) selects specific columns from the relation.
 - * *Rename* changes the name of the relation or its attributes.
 - **Binary operations:** These require two relations.
 - * *Union* combines rows from two relations.

- * *Set difference* finds rows in one relation but not the other.
- * *Intersection* finds common rows between two relations.
- * *Cartesian product* pairs each row of one relation with every row of another.
- * *Join* combines rows from two relations based on a related attribute.

- **Selection Σ :** This operation is used to filter data. For example, if you want to find all instructors in the Physics department, you would use a selection operation to filter out only those rows where the department name is “Physics”.
- **Projection π :** This operation is used to narrow down the columns of interest. For instance, if you only need the names and salaries of instructors, you would project these two attributes from the instructor relation.
- **Set operations:** These operations allow you to combine or compare two relations. It’s important that the relations involved have the same attributes to ensure compatibility. This is similar to how you might combine or compare sets in mathematics.

11 / 13: Relational Algebra: 2/4

Relational Algebra: 2/4

- **Cartesian product:** combine two relations into a new one
 - instructor = (ID, name, dept_name, salary)
 - teaches = (ID, course_id, sec_id, semester, year)
- E.g., instructor \times teaches gives (instructor.ID, instructor.name, instructor.dept_name, teaches.ID, ...)

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	Califleri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

instructor relation

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

teaches relation

instructor.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...

instructor \times teaches



- **Cartesian product:** This operation is a fundamental concept in relational algebra, which is a part of database theory. It allows us to combine two tables (or relations) into a single new table. The new table contains all possible combinations of rows from the original tables.
 - In this example, we have two relations: **instructor** and **teaches**. The **instructor** relation includes columns for ID, name, dept_name, and salary. The **teaches** relation includes columns for ID, course_id, sec_id, semester, and year.
- When we perform a Cartesian product of **instructor** and **teaches**, we create a new relation that includes every possible pairing of rows from these two tables. This means that for

each row in the `instructor` table, it is paired with every row in the `teaches` table. The resulting table will have columns from both tables, such as `instructor.ID`, `instructor.name`, `instructor.dept_name`, `teaches.ID`, and so on.

- **Visual aids:** The images provided in the slide show the `instructor` and `teaches` relations separately, and then the result of their Cartesian product. This visual representation helps in understanding how the Cartesian product operation combines the data from both tables into a larger set of data.
- *Important note:* While the Cartesian product is a powerful tool, it can result in very large tables, especially if the original tables have many rows. This is because the number of rows in the resulting table is the product of the number of rows in the original tables.

12 / 13: Relational Algebra: 3/4

Relational Algebra: 3/4

- **Join:** composition of two operations
 - Cartesian product
 - Selection based on equality between two fields
 - E.g., `instructor × teaches` when `instructor.ID = teaches.ID`

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	Califleri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

`instructor` relation

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

`teaches` relation

instructor.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-319	2	Spring	2018
98345	Kim	Elec. Eng.	80000	98345	EE-181	1	Spring	2017

$\sigma_{\text{instructor.ID} = \text{teaches.ID}}(\text{instructor} \times \text{teaches})$

- **Join:** composition of two operations
 - The *join* operation in relational algebra is a fundamental concept used to combine two tables based on a related column. It is essentially a combination of two operations: the Cartesian product and selection.
 - **Cartesian product:** This operation takes two tables and pairs every row from the first table with every row from the second table. While this creates a large number of combinations, it is not very useful on its own because it doesn't consider any relationships between the data.
 - **Selection based on equality between two fields:** After the Cartesian product, the selection operation is applied to filter the results. This selection is based on a condition, typically an equality between two fields from the different tables. For example, if we have an `instructor` table and a `teaches` table, we might join them on the condition that

`instructor.ID = teaches.ID`. This means we only keep the rows where the instructor's ID matches the ID in the teaches table, effectively linking instructors to the courses they teach.

- The images in the slide likely show examples of the `instructor` and `teaches` relations, as well as the result of the join operation, illustrating how the data is combined based on the specified condition.

13 / 13: Relational Algebra: 4/4

Relational Algebra: 4/4

- **Query:** combination of relational algebra operations
 - E.g., "find `course_id` from table `section` for fall 2017"
- **Assignment:** assign parts of relational algebra to temporary relation variables
 - Write a query as a sequential program
 - E.g., "find `course_id` for classes in both fall 2017 and spring 2018"
- **Equivalent queries:** two queries giving the same result on any DB instance
 - Some formulations are more efficient

$\Pi_{\text{course_id}}(\sigma_{\text{semester} = \text{"Fall"} \wedge \text{year} = 2017}(\text{section}))$

$\begin{aligned} \text{courses_fall_2017} &\leftarrow \Pi_{\text{course_id}}(\sigma_{\text{semester} = \text{"Fall"} \wedge \text{year} = 2017}(\text{section})) \\ \text{courses_spring_2018} &\leftarrow \Pi_{\text{course_id}}(\sigma_{\text{semester} = \text{"Spring"} \wedge \text{year} = 2018}(\text{section})) \\ \text{courses_fall_2017} &\cap \text{courses_spring_2018} \end{aligned}$



13 / 13

- **Query:** In relational algebra, a query is essentially a combination of operations that allow us to retrieve specific data from a database. For example, if you want to find the `course_id` from a table named `section` for the fall of 2017, you would use a series of operations to filter and select the relevant data. This is similar to asking a question to the database and getting the answer in the form of data.
- **Assignment:** Sometimes, it's useful to break down complex queries into smaller parts. This is where assignment comes in. You can assign parts of your query to temporary variables, making it easier to manage and understand. Think of it like writing a step-by-step program where each step builds on the previous one. For instance, if you want to find `course_id` for classes in both fall 2017 and spring 2018, you might first find each separately and then combine the results.
- **Equivalent queries:** In relational algebra, different queries can sometimes produce the same result. These are known as equivalent queries. However, not all equivalent queries are created equal—some are more efficient than others. This means they can retrieve the same data faster or with less computational effort, which is important for optimizing database performance.