Week 2 Lecture 1

Class	BSCCS2001
Created	@August 21, 2021 12:32 PM
Materials	
Module #	6
Type	Lecture
Week #	2

Introduction to Relational Model

Attribute Types

- Consider

  Student = Roll #, First Name, Last Name, DoB, Passport #, Aadhaar #, Department

  relation

- The set of allowed values for each attribute is called the **domain** of the attribute
  - **Roll #** - Alphanumeric string
  - **First Name, Last Name** - Alpha string
  - **DoB** - Date
  - **Passport #** - String (Letter followed by 7 digits) - nullable (Optional)
  - **Aadhaar #** - 12-digit number
  - **Department** - Alpha string
- Attribute values are (normally) required to be **atomic**; this is, indivisible
- The special value **null** is a member of every domain. Indicates that the value is *unknown*
- the *null* value may cause complications in the definition of many operations

Roll #	First Name	Last Name	DoB	Passport	Aadhaar	Dept.
15CS10026	Lalit	Dubey	27-Mar-1997	L4032464	172861749239	Computer

Roll #	First Name	Last Name	DoB	Passport	Aadhaar	Dept.
16EE30029	Jatin	Chopra	17-Nov-1996	null	391718363816	Electrical

Relational Schema and Instance

- $A_1, A_2, ..., A_n$ are the attributes
- R = $(A_1, A_2, ..., A_n)$ is a relation schema

  Example: `instructor = (ID, name, dept_name, salary)`
- Formally, given as $D_1, D_2, ..., D_n$ a relation r is a subset of

  $D_1 \times D_2 \times ... D_n$

  Thus, a relation is a set of n-tuples $(a_1, a_2, ..., a_n)$ where each $a_i \in D_i$
- The current values (**relation instance**) of a relation are specified by a table
- An element t or r is a tuple, represented by a row in a table
- Example

  instructor ≡ (String(5) × String × String × Number+), where ID ∈ String(5), name ∈ String, dept_name ∈ String and salary ∈ Number+

Keys

- Let $K \subseteq R$, where R is the set of attributes in the relation
- K is a **superkey** of R if values of 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**
- A **surrogate key** (or synthetic key) in a database is a unique identifier for either an entity in the modeled world or an object in the database
  - The surrogate key is not derived from application data, unlike a natural (or business) key which is derived from application data

Keys: Examples

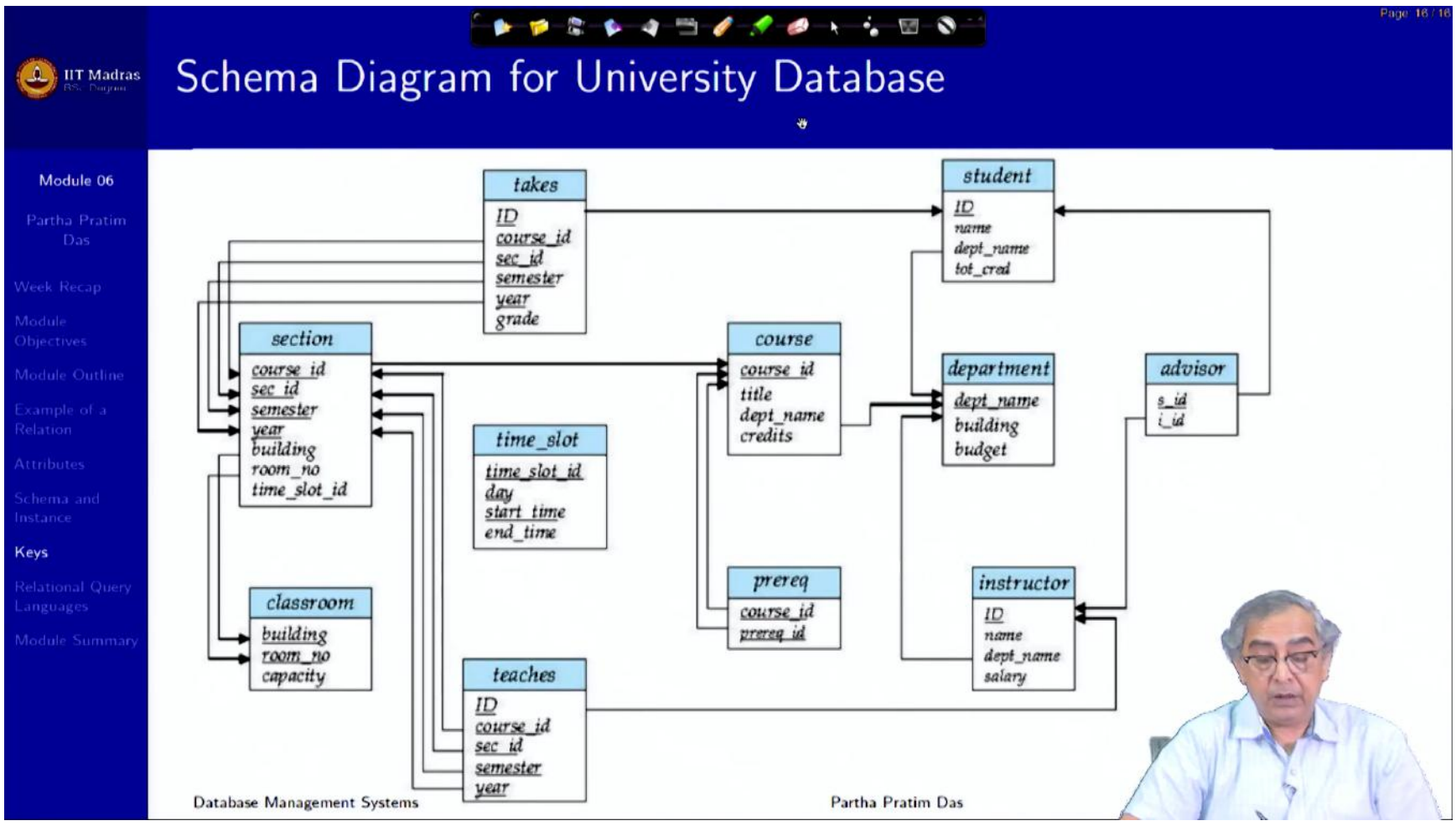
- Students = Roll #, First Name, Last Name, DoB, Passport #, Aadhaar #, Department
- **Super Key**: Roll #, {Roll #, DoB}
- **Candidate Keys**: Roll #, {First Name, Last Name}, Aadhaar #
  - Passport # cannot be a key because it is an optional field and can take null values, but an ID can never be null
- **Primary Key**: Roll #
  - Can Aadhaar # be a key?
  - It may suffice for unique identification, but Roll # may have additional useful information.
  - For example: 14CS92P01
    - Read it as 14-CS-92-P-01
    - 14 - Admission in 2014
    - CS - Department: Computer Science
    - 92 - Category of the Student
    - P - Type of admission: *Project*
    - 01 - Serial Number
- **Secondary / Alternate Key**: {First Name, Last Name}, Aadhaar #
- **Simple Key**: Consists of a single attribute

- **Composite Key**: {First Name, Last Name}
  - Consists of more than one attribute to uniquely identify an entity occurrence
  - One or more of the attributes, which make up the key are not simple keys in their own right

Roll #	First Name	Last Name	DoB	Passport	Aadhaar	Dept
15CS10026	Lalit	Dubey	27-Mar-1997	L4032464	172861749239	Computer
16EE30029	Jatin	Chopra	17-Nov-1996	null	391718363816	Electrical
15EC10016	Smriti	Mongra	23-Dec-1996	G5432849	204592710914	Electronics
16CE10038	Dipti	Dutta	02-Feb-1997	null	571919482918	Civil
15CS30021	Ramdin	Minz	10-Jan-1997	X8811623	492849275924	Computer

- **Foreign key constraint**: Value in one relation must appear in another (in other words, when a particular attribute is a key in a different table)
  - **Referencing** relation
    - Enrolment: Foreign Keys - Roll #, Course #
  - **Referenced** relation
    - Students, Courses
- A **compound key** consists of more than one attribute to uniquely identify an entity occurence
  - Each attribute, which makes up the key, is a simple key in its own right
  - {Roll #, Course #}

Schema Diagram for University Database



Relational Query Languages

Procedural viz-a-viz Non-procedural or Declarative Paradigms

- Procedural programming requires that the programmer tell the computer what to do
  - That is, how to get the output for the range of required inputs
  - The programmer must know an appropriate algorithm
- Declarative programming requires a more descriptive style
  - The programmer must know what relationships hold between various entities

Relational Query Language: Example

Procedural vs. Non-procedural or Declarative Paradigms

- **Example: Square root of n**
  - Procedural
    a) Guess x_0 (close to root of n)
    b) $i \leftarrow 0$
    c) $x_{i+1} \leftarrow (x_i + n/x_i)/2$
    d) Repeat Step 2 if $|x_{i+1} - x_i| > delta$
  - Declarative
    - ▷ Root of n is m such that $m^2 = n$

- "Pure" languages:
  - Relational Algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate on relational algebra
  - Not Turing-macine equivalent
    - Not all algorithms can be expressed in Relational Algebra
  - Consists of 6 basic operations

Week 2 Lecture 2

Class	BSCCS2001
Created	@August 22, 2021 6:57 PM
Materials	https://www.caam.rice.edu/~heinken/latex/symbols.pdf
Module #	7
Type	Lecture
Week #	2

Introduction to Relational Model (part 2)

Relational Operators

Basic properties of relations

- A relation is a set. Hence,
- Ordering of rows / tuples is inconsequential
- All rows / tuples must be distinct

Select operation - selection of rows (tuples)

- Relation r on the following table

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

- The select operation is defined as

$$\sigma_{A=B \wedge D>5}(r)$$

- And it returns the following table as a result

A	B	C	D
α	α	1	7
β	β	23	10

Project operation - selection of columns (Attributes)

- Relation r

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- The projection operation is defined as

$$\pi_{A,C}(r)$$

- And it returns the following table as a result

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Partha Pratim Das

Union of two relations

- Relation r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- The union of two relation is defined as

$$r \cup s$$

- And it returns the following result

A	B
α	1
α	2
β	1
β	3

Set difference of two relations

- Relation r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- The set difference of two relations is defined as

$$r - s$$

- And it returns the following result

A	B
α	1
β	1

Note: $r \cap s = r - (r - s)$

Joining two relations - Cartesian-product

- Relation 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

- The cartesian product is defined as

$$r \times s$$

- And it returns the following result

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

Cartesian-product - Naming issue

A	B
α	1
β	2

r

B	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

A	r.B	s.B	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

Renaming a Table

- Allows us to refer to a relation, say E, by more than one name

$$\rho_X(E)$$

returns the expression E under the name X

- Relations r

A	B
α	1
β	2

r

- Self product

$$r \times \rho_s(r)$$

r.A	r.B	s.A	s.B
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2

Composition of Operations

- Can build expressions using multiple operations
- Example:

$$\sigma_{A=C}(r \times s)$$

- $r \times s$

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

$$\sigma_{A=C}(r \times s)$$

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

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 $R \cup S$
  - 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

$$r \bowtie s$$

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

$$\pi_{A,r.B,C,r.D,E}\left(\sigma_{r.B=s.B \,\wedge\, r.D=s.D}\left(r \times s\right)\right)$$

Aggregation Operators

- Can we compute:
  - SUM
  - AVG
  - MAX
  - MIN

Notes about Relational Languages

- Each query input is a table (or a set of tables)
- Each query output is a table
- All data in the output table appears in one of the input tables
- Relational Algebra is not Turing complete

Week 2 Lecture 3

Class	BSCCS2001
Created	@August 22, 2021 8:38 PM
Materials	
Module #	8
Type	Lecture
Week #	2

Introduction to Structured Query Language (SQL)

History of SQL

- IBM developed *Structured English Query Language* (**SEQUEL**) as a part of System R project.
- Renamed Structured Query Language (SQL: *still pronounced as* **SEQUEL**)

ANSI and ISO standard SQL:

Aa Name	Description
SQL - 86	First formalized by **ANSI**
SQL - 89	+ **Integrity Constraints**
SQL - 92	Major revision (**ISO/IEC 9075 standard**), **De-facto Industry Standard**
SQL : 1999	+ **Regular Expression Matching, Recursive Queries, Triggers, Support for Procedural and Control Flow Statements**, Non-scalar types (Arrays) and some OO features (structured types), **Embedding SQL in Java (SQL/OLB)** and **Embedding Java in SQL (SQL/JRT)**
SQL : 2003	+ **XML features (SQL/XML)**, Window functions, Standardized sequences and columns with auto-generated values (identity columns)
SQL : 2006	+ Way of **importing and storing XML data** in a SQL database, **manipulating it** within the database, and **publishing both XML and conventional SQL-data in XML form**
SQL : 2008	Legalizes **ORDER BY** outside Cursor Definitions + **INSTEAD OF** Triggers, **TRUNCATE** statements and **FETCH** clause

Aa Name	Description
SQL : 2011	+ Temporal data (**PERIOD FOR**) Enhancements for Window functions and **FETCH** clause
SQL : 2016	+ Row Pattern Matching, Polymorphic Table Functions and **JSON**
SQL : 2019	+ Multidimensional Arrays (MDarray type and operators)

Compliance

- SQL is the de facto industry standard today for relational or structured data systems
- Commercial system as well as open system may be fully or partially compliant to one or more standards from SQL-92 onward
    - Not all examples here may work on your particular system. Check your system's SQL docs.

Alternatives

- There aren't any alternatives to SQL for speaking to relational databases (i.e. SQL as a protocol)
    - There are alternatives to writing SQL in the applicaions
- These alternatives have been implemented in the form of front-ends for working with relational databases. Some examples of a front-end include (for a section of languages):
    - **SchemeQL** and **CLSQL**
        - Probably the most flexible, thanks to their Lisp heritage
        - They also look a lot more like SQL than other front-ends
    - **LINQ** (in .NET)
    - **ScalaQL** and **ScalaQuery** (in Scala)
    - **SqlStatement**, **ActiveRecord** and many others in Ruby
    - **HaskellDB**
    - ... the list goes on for many other languages

Derivatives

- There are several query languages that are derived from or inspired by SQL.
- Out of these, the most popular and effective is **SPARQL**.
- **SPARQL** (pronounced *sparkle,* a recursive acronym for *SPARQL Protocol and RDF Query Language*) is an RDF query language
    - A semantic query language for databases - able to retrieve and manipulate data stored in ***Resource Description Framework*** (***RDF***) format.
    - It has been standardized by the W3C Consortium as key technology of the semantic web
    - Versions
        - SPARQL 1.0 (Jan. 2008)
        - SPARQL 1.1 (Mar. 2013)
    - Used as the query languages for several NoSQL systems - particularly the Graph Databases that use RDF as store

Data Definition Language (DDL)

The SQL data-definition language (DDL) allows the specification of information about relations, including:

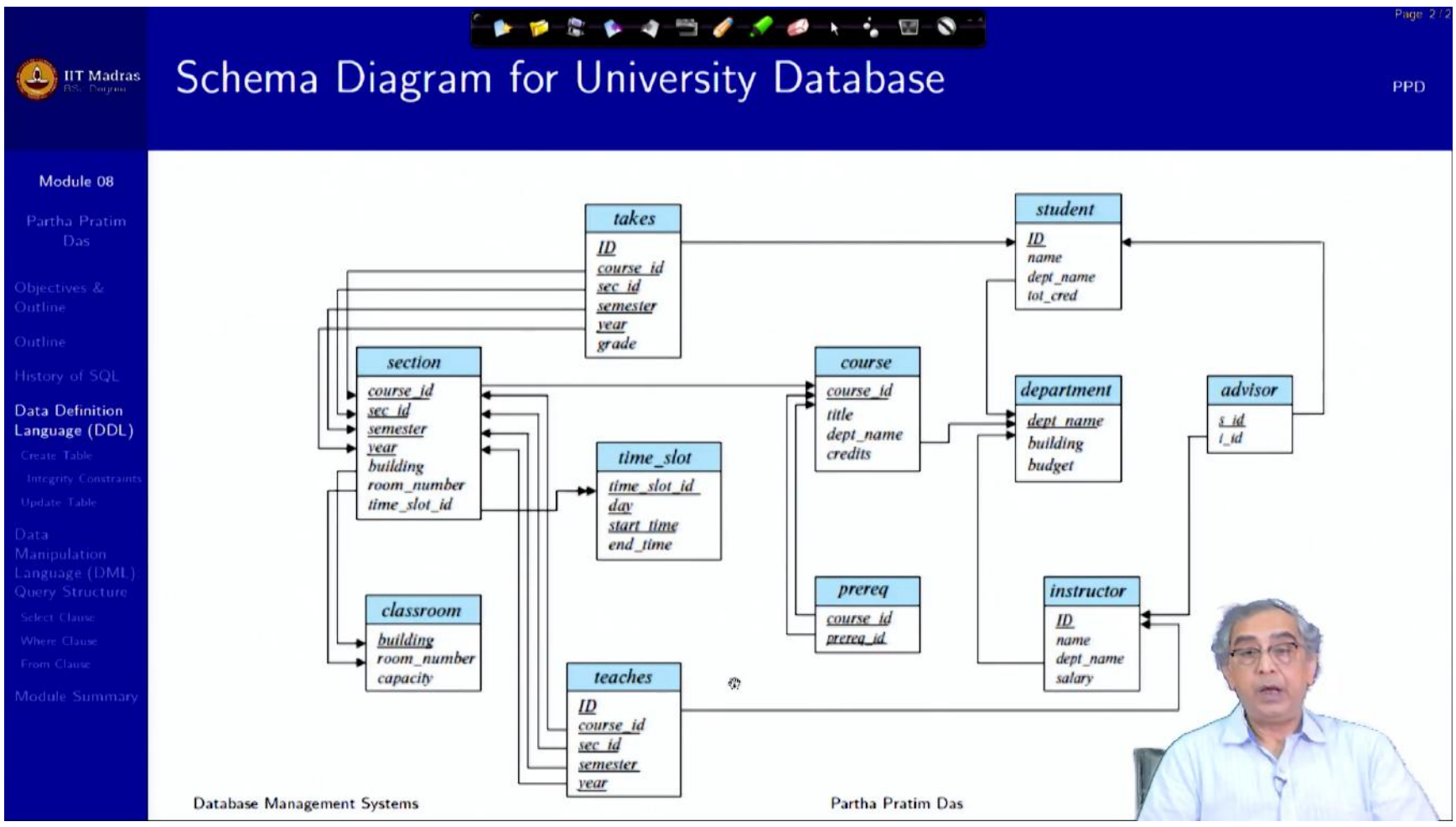
- The *Schema* for each *Relation*
- The *Domain* of values associated with each *Attribute*
- *Integrity Constraints*

- And, as we will see later, also other information such as ...
  - The set of *Indices* to be maintained for each relations
  - *Security and Authorization* information for each relation
  - The *Physical Storage Structure* of each relation on disk

Domain types (or Data types) in SQL

- ***char(n)*** - Fixed length character string, with user-specified length *n*
- ***varchar(n)*** - Variable length character strings, with user-specified max length *n*
- ***int*** - Integer (a finite subset of the integers that is machine-dependent)
- ***smallint(n)*** - Small integer (a machine-dependent subset of the integer domain type)
- ***numeric(p, d)*** - Fixed point number, with user-specified precision of *p* digits, with *d* digits to the right of decimal point. (ex. *numeric(3, 1)* allows 44.5 to be stored exactly, but not 444.5 or 0.32)
- ***real, double precision*** - Floating point and double-precision floating point numbers, with machine-dependent precision
- ***float(n)*** - Floating point number with user specified precision of at-least *n* digits

Schema diagram for a University database



Create Table construct

- An SQL relation is defined using the **create table** command:

  create table *r* $(A_1 D_1, A_2 D_2, ..., A_n D_n)$,

  $(integrity - constraint_1)$,

  ...

  $(integrity - constraint_k))$;

- ***r*** is the name of the relation (table)
- each A_i is an attribute name in the schema of relation ***r***
- D_i is the data type of values in the domain of attribute A_i

Example

```
create table instructor (
  ID char(5),
```

```
  name varchar(20),
  dept_name varchar(20),
  salary numeric(8, 2));
```

University DB

Aa instructor
ID
name
dept_name
salary

Create Table constructs: Integrity constraints

- **not null**
- **primary key** $(A_1, ..., A_n)$
- **foreign key** $(A_m, ..., A_n)$ **references *r***

```
create table instructor (
  ID char(5),
  name varchar(20),
  dept_name varchar(20),
  salary numeric(8, 2));
```

```
create table instructor (
  ID char(5),
  name varchar(20) not null,
  dept_name varchar(20),
  salary numeric(8, 2),
  primary key (ID),
  foreign key (dept_name) references department));
```

primary key declaration on an attribute automatically ensures ***not null***

Create Table construct: More relations

```
create table student (
  ID varchar(5),
  name varchar(20) not null,
  dept_name varchar(20),
  tot_cred numeric(3, 0),
  primary key (ID),
  foreign key (dept_name) references department);
```

```
create table course (
  course_id varchar(8),
  title varchar(50),
  dept_name varchar(20),
  credits numeric(2, 0),
  primary key (course_id),
  foreign key (dept_name) references department);
```

```
create table takes (
  ID varchar(5),
  course_id varchar(8),
  sec_id varchar(8),
  semester varchar(6),
  year numeric(4, 0),
  grade varchar(2),
  primary key (ID, course_id, sec_id, semester, year),
  foreign key (course_id, sec_id, semester, year) references section);
```

- **NOTE:** *sec_id* can be dropped from primary key above to ensure a student cannot register for two sections of the same course in the same semester

Update Tables

- **Insert** (DML command)

```
insert into instructor values ('10211', 'Smith', 'Biology', 66000);
```

- **Delete** (DML command)
  - Remove all tuples from the *student* relation

```
delete from student
```

- **Drop Table** (DDL command)

```
drop table r
```

- **Alter** (DDL command) # to edit the schema

```
alter table r add A D
```

  - Where *A* is the name of the attribute to be added to relation to *r* and *D* is the domain of *A*
  - All existing tuples in the relation are assigned ***null*** as the value for the new attribute

```
alter table r drop A
```

  - Where *A* is the name of the attribute of relation *r*
  - Dropping of attributes not supported by many databases

Data Manipulation Language (DML): Query Structure

Basic query structure

- A typical SQL query has the form:

  select $A_1, A_2, ..., A_n,$

  from $r_1, r_2, ..., r_m$

  where *P*

  - A_i represents an attribute from r_i's
  - r_i represents a relation
  - *P* is a predicate
- The result of an SQL query is a relation

SELECT clause

- The **select** clause lists the attributes desired in the result of a query
  - Corresponds to the projection operation of relational algebra
- Example: find the names of all instructors

```
select name from instructor
```

- **NOTE:** SQL names are case insensitive
  - Name = NAME = name
  - Some people prefer to use UPPER CASE wherever we use the **bold font**
- **SQL allows duplicates in relations as well as in query results**

- To force the elimination of duplicates, insert the keyword **distinct** after **select**
- Find the department names of all instructors and remove duplicates

```
select distinct dept_name
from instructor
```

- The keyword **all** specifies that duplicates should not be removed

```
select all dept_name
from instructor
```

- An asterisk (*) in the select denotes *all attributes*

```
select *
from instructor
```

- An attribute can be a literal with no **from** clause

```
select '437'
```

  - Result is a table with one column and a single row with the value '437'
  - Can give the column a name using:

```
select '437' as FOO
```

- An attribute can be a literal with **from** clause

```
select 'A'
from instructor
```

  - Result is a table with one column and N rows (number of tuples in the *instructors* table), each row with value 'A'

The **select** clause can contain arithmetic expressions involving the operation +, -, * and / and operating on constants or attributes of tuples

- The query:

```
select ID, name, salary/12
from instructor
```

- Would return a relation that is the same as the *instructor* relation, except that the value of the attribute *salary* is divided by 12
- Can rename "*salary*/12" using the **as** clause:

```
select ID, name, salary/12 as monthly_salary
```

WHERE clause

- The **where** clause specifies conditions that the result must satisfy
  - Corresponds to the selection predicate of the relational algebra
- To find all instructors in the Computer Science department

```
select name
from instructor
where dept_name = 'Comp. Sci.'
```

- Comparison results can be combined using the logical connectives **and, or, not**

- To find all instructors in Comp. Sci. department with salary > 80000

```
select name
from instructor
where dept_name = 'Comp. Sci.' and salary > 80000
```

- Comparisons can be applied to results of arithmetic expressions

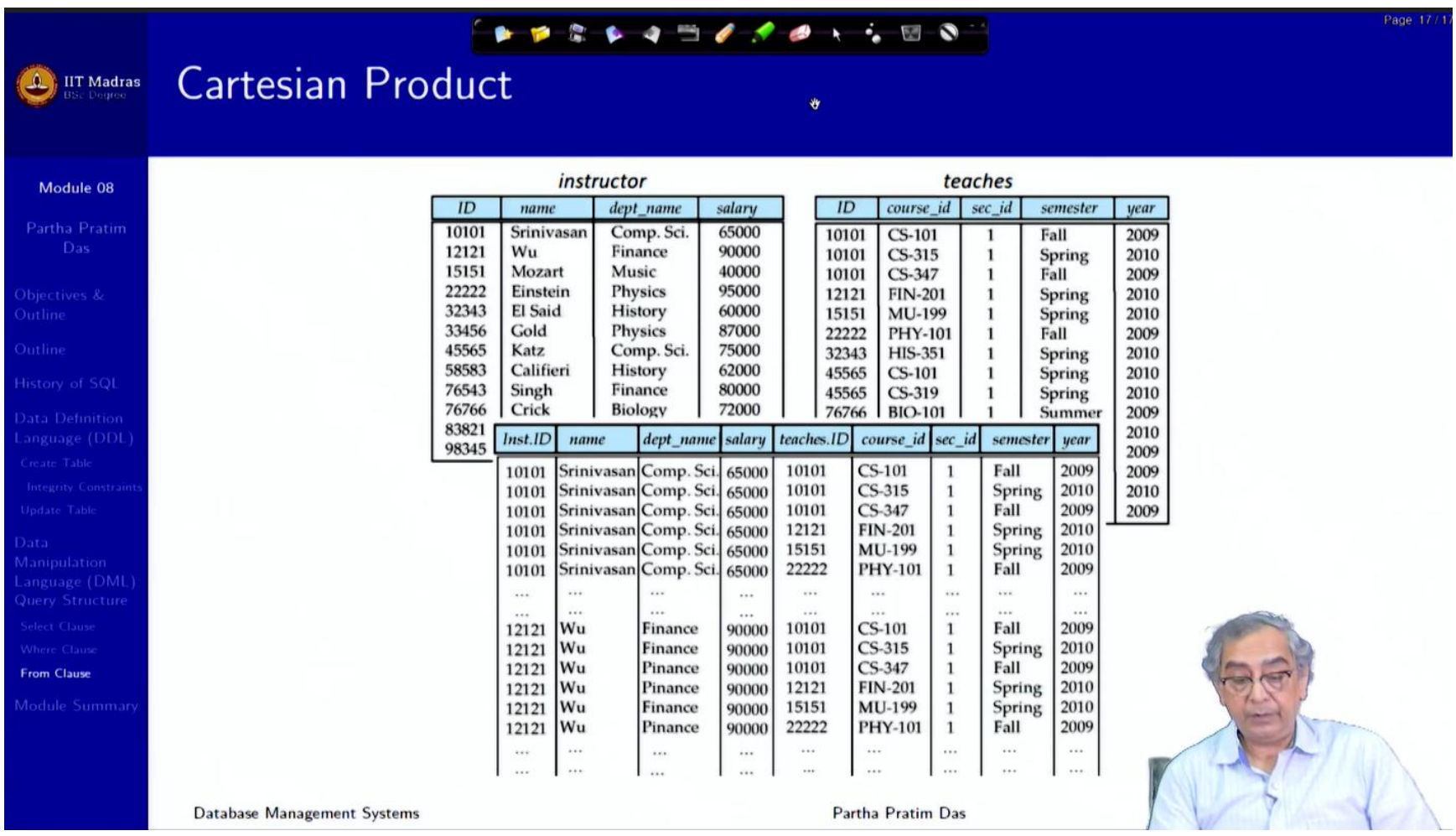
FROM clause

- The **from** clause lists the relations involved in the query
  - Corresponds to the Cartesian product operation of the relational algebra
- Find the Cartesian product *instructor X teaches*

```
select *
from instructor, teaches
```

  - Generates every possible instructor-teaches pair with all attributes from both relations
  - For common attributes (for eg: ID), the attributes in the resulting table are renamed using the relation name (for eg: instructor.ID)
- Cartesian product is not very useful directly, but useful when combined with the where-clause condition (selection operation in relational algebra)

Cartesian product



Week 2 Lecture 4

Class	BSCCS2001
Created	@September 3, 2021 11:26 AM
Materials	
Module #	9
Type	Lecture
Week #	2

Introduction to Structured Query Language (SQL) (part 2)

Cartesian product (cont. from the previous lecture's end)

Example

- Find the names of all instructors who have taught some courses and the course_id

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID
```

- Equi-Join, Natural Join

instructor

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			
98345			

teaches

ID	*course_id*	*sec_id*	*semester*	*year*
10101	CS-101	1	Fall	2009
10101	CS-315	1	Spring	2010
10101	CS-347	1	Fall	2009
12121	FIN-201	1	Spring	2010
15151	MU-199	1	Spring	2010
22222	PHY-101	1	Fall	2009
32343	HIS-351	1	Spring	2010
45565	CS-101	1	Spring	2010
45565	CS-319	1	Spring	2010
76766	BIO-101	1	Summer	2009
				2010
				2009
				2009
				2010
				2009

Inst.ID	*name*	*dept_name*	*salary*	*teaches.ID*	*course_id*	*sec_id*	*semester*	*year*
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2009
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2010
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2009
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2010
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2009
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2010
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2010
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2009
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

- Here in this table, we do not have the names of the courses
- If we want the name, we will again have to do a similar join operation with a table that has the names of the courses
  - This operations is known as **Natural Join**
- Example

  Find the names of all the instructors in the Art dept. who have taught some courses and the *course_id*

```
select name, course_id
from instructor, teaches
where instructor.ID = teaches.ID and instructor.dept_name = 'Art'
```

Rename AS operation

- The SQL allows renaming relations and attributes using the **as** clause:

```
old_name as new_name
```

- Find the names of all the instructors who have a higher salary than some instructor in 'Comp. Sci.'

```
select distinct T.name
from instructor as T, instructor as S
where T.salary > S.salary and S.dept_name = 'Comp. Sci.'
```

- The keyword **as** is optional and may be omitted

  instructor **as** $T \equiv$ *instructor T*

String Operations

- SQL includes a string-matching operator for comparisons on character strings.
- The operator **like** uses patterns that are described using two special characters:
  - percent (%)

    The % character matches any sub-string

    - underscore (_)

      The _ character matches any character

- Find the names of all instructors whose name includes the sub-string "dar"

```
select name
from instructor
where name like '%dar%'
```

- Match the string "100%"

```
like '100%' escape '\'
```

  in the above example, we use the backslash (\ ) as the escape character

  and **'%dar%'** could match **Dar**win, Majum**dar,** Sar**dar** or Ud**dar**in

  meanwhile, '%dar___' (**dar** followed by 3 underscores), it will match **Dar**win, but not the others

- Patterns are *case sensitive*
- Pattern matching example
  - 'Intro%' matches any string beginning with "Intro"
  - '%Comp%' matches any string containing "Comp" as a substring
  - '___' (3 underscores) many any string of exactly 3 characters
  - '___%' (3 underscores and then a %) matches any string of at least 3 characters
- SQL supports variety of string operations such as
  - Concatenation (using "||") [double pipe symbol]
  - Converting from upper to lower case (and vice-versa)
  - Finding the string length, extracting substrings, etc...

Ordering the display of tuples (ORDER BY clause)

- List in alphabetic order the names of all the instructors

```
select distinct name
from instructor
order by name
```

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default
  - Example: **order by** *name* **desc**
- Can sort on multiple attributes
  - Example: **order by** *dept_name*, *name*

Selecting number of tuples in output

- The **Select Top** clause is used to specify the number of records to return
- The **Select Top** clause is useful on large tables with thousands of records.
  - Returning a large number of records can impact performance

```
select top 10 distinct name
from instructor
```

- Not all database systems support the **SELECT TOP** clause.
  - SQL Server & MS Access support **select top**
  - MySQL supports the **limit** clause

- Oracle uses **fetch first** *n* **rows only** and **rownum**

```
select distinct name
from instructor
order by name
fetch first 10 rows only
```

WHERE clause predicates

- SQL includes a **between** comparison operator
- Example: Find the names of all the instructors with salary between $90,000 and $100,000

  (that is, $\geq$ $90,000 and $\leq$ $100,000)

```
select name
from instructor
where salary between 90000 and 100000
```

- Tuple comparison

```
select name, course_id
from instructor, teaches
where (instructor.ID, dept_name) = (teaches.ID, 'Biology');
```

IN operator

- The **in** operator allows you to specify multiple values in a **where** clause
- The **in** operator is a shorthand for multiple **or** conditions

```
select name
from instructor
where dept_name in ('Comp. Sci.', 'Biology')
```

Duplicates

- In relations with duplicates, SQL can define how many copies of tuples appear in the result
- **Multiset** versions of some of the relational algebra operators - given multiset relations r_1 and r_2:

  a) **SELECT** $\sigma_\theta(r_1)$ **:** If there are c_1 copies of tuple t_1 in r_1 and t_1 satisfies selection σ_θ, then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$

  b) **PROJECTION** $\Pi_A(r)$ **:** For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$ where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1

  c) $r_1 \times r_2$ **:** If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuples t_2 in r_2, there are $c_1 \times c_2$ copies of the tuple $t_1 \cdot t_2$ in $r_1 \times r_2$
- Example: Suppose multiset relations $r_1(A, B)$ and $r_2(C)$ are as follows:

  $r_1 = \{(1, a)(2, a)\}$; $r_2 = \{(2), (3), (3)\}$
- Then $\Pi_B(r_1)$ would be $\{(a), (a)\}$ while $\Pi_B(r_1) \times r_2$ would be

  $\{(a, 2), (a, 2), (a, 3), (a, 3), (a, 3), (a, 3)\}$
- SQL duplicate semantics:

  select $A_1, A_2, ..., A_n$

  from $r_1, r_2, ..., r_m$

  where *P*

  is equivalent to the multiset version of the expression:

  $$\Pi_{A_1, A_2, ..., A_n}(\sigma_P(r_1 \times r_2 \times ... \times r_m))$$

Week 2 Lecture 5

Class	BSCCS2001
Created	@September 4, 2021 6:05 PM
Materials	
Module #	10
Type	Lecture
Week #	2

Introduction to Structured Query Language (SQL) (part 3)

Set operations

Example

- Find the courses that ran in Fall 2009 **or** in Spring 2010

```
(select course_id from section where sem = 'Fall' and year = 2009)
union
(select course_id from section where sem = 'Spring' and year = 2010)
```

- Find the courses that ran in Fall 2009 **and** in Spring 2010

```
(select course_id from section where sem = 'Fall' and year = 2009)
intersect
(select course_id from section where sem = 'Spring' and year = 2010)
```

- Find the courses that ran in Fall 2009 **but not** in Spring 2010

```
(select course_id from section where sem = 'Fall' and year = 2009)
except
(select course_id from section where sem = 'Spring' and year = 2010)
```

- Find the salaries of all the instructors that are less than the largest salary

```
select distinct T.salary
from instructor as T, instructor as S
where T.salary < S.salary
```

- Find the salaries of all the instructors

```
select distinct salary
from instructor
```

- Find the largest salary of all the instructors

```
(select distinct salary from instructor)
except
(select distinct T.salary from instructor as T, instructor as S where T.salary < S.salary)
```

- Set operations such as **union**, **intersect** and **except** automatically eliminate the duplicates
- To retain all the duplicates, use the corresponding multiset versions **union all**, **intersect all** and **except all**
- Suppose a tuple occurs *m* times in *r* and *n* times in *s*, then it occurs ...
  - $m + n$ times in *r* **union all** *s*
  - min(*m*, *n*) times in *r* **intersect all** *s*
  - max(0, *m* - *n*) times in *r* **except all** *s*

NULL values

- What is a NULL value?

  A NULL value is something unknown or a value that does not exist yet
- Why is NULL value so important?
  - Certain values may not exist for everyone

    For eg: Every student may not have a passport at the time of registration
  - Often times while we are creating/inserting a record, we may not know all the values of all the fields

    For eg: When a student joins, the student does not have any credit assigned to him/her, so the total credit is NULL

    We can say 0 (zero), but 0 (zero) and NULL are different

    0 (zero) means the student has not taken a credit

    NULL means the credit has not been given yet
  - Naturally, when we add an attribute to all the existing rows of a table, the value of the particular field cannot be known, cannot be set, so it will have to initialized as a NULL value
- It is possible for tuples to have a *null* value, denoted by **null**, for some of their attributes
- The predicate **is null** can be used to check for *null* values
  - Example: Find all the instructors whose salary is *null*

```
select name
from instructor
where salary is null
```

- It is not possible to test for *null* values with comparison operators such as $=, <, > or <>$

  We need to use the **is null** and **is not null** operators instead

NULL values: Three valued logic

- Three values - **true, false, unknown**
- Any comparison with *null* returns *unknown*
  - Example: $5 < null$ or $null <> null$ or $null = null$

- Three-valued logic using the value unknown:
  - **OR:**

    (*unknown* **or** *true*) = *true*

    (*unknown* **or** *false*) = *unknown*

    (*unknown* **or** *unknown*) = *unknown*
  - **AND:**

    (*true* **and** *unknown*) = *unknown*

    (*false* **and** *unknown*) = *false*

    (*unknown* **and** *unknown*) = *unknown*
  - **NOT:**

    (**not** *unknown*) = *unknown*
  - "*P* **is unknown**" evaluates to *true* if predicate *P* evaluates to *unknown*
- Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

Aggregate functions

- These functions operate on the multiset of values of a column of a relation (table) and return a value

  avg: average value

  min: minimum value

  max: maximum value

  sum: sum of the values

  count: number of values

Examples

- Find the average salary of instructors in the Computer Science department

```
select avg(salary)
from instructor
where dept_name = 'Comp. Sci.'
```

- Find the total number of instructors who teach a course in the Spring 2010 semester

```
select count(distinct ID)
from teaches
where semester = 'Spring' and year = 2010
```

- Find the number of tuples in the *course* relation (table)

```
select count(*)
from courses;
```

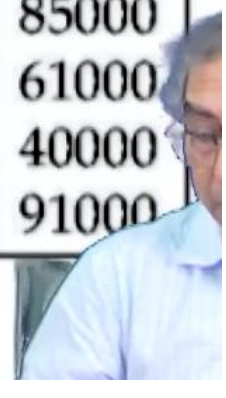
Example (GROUP BY)

- Find the average salary of instructors in each department

```
select dept_name. avg(salary) as avg_salary
from instructor
group by dept_name;
```

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

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000



So, **group by** takes a column and makes sub-tables of all those records which have the same value on that particular group by attribute

It then applies the aggregate function on the column based on this sub-table

- Attributes in **select** clause outside of aggregate functions must appear in **group by** list

```
-- The following query is incorrect because of the 'ID' attribute
select dept_name, ID, avg(salary)
from instructor
group by dept_name;
```

HAVING clause

- Find the names and average salaries of all departments whose average salary is greater than 42,000

```
select dept_name, ID, avg(salary)
from instructor
group by dept_name
having avg(salary) > 42000;
```

NOTE: Predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

NULL values and aggregates

- Total all salaries

```
select sum(salary)
from instructor;
```

  - Above statement ignores null amounts
  - Result is *null* if there is no non-null amount
- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0 (zero)
  - all other aggregates return null