Intro to Databases (COMP_SCI 339)

Course Introduction



TODAY'S AGENDA

Relational Model Relational Algebra SQL Overview

COURSE OVERVIEW

This course is about the design and implementation of database management systems (DBMSs).

This is **not** a course about how to use a DBMS to build applications or how to administer a DBMS.

DATABASE EXAMPLE

Create a database that models a digital music store to keep track of artists and albums.

Things we need for our store:

- → Information about Artists
- → What <u>Albums</u> those Artists released

FLAT FILE STRAWMAN

Store our database as comma-separated value (CSV) files that we manage ourselves via application code.

- → Use a separate file per entity.
- → The application must parse the files each time we want to read/update records.

FLAT FILE STRAWMAN

Create a database that models a digital music store.

Artist(name, year, country)

"Wu-Tang Clan",1992,"USA"

"Notorious BIG",1992,"USA"

"GZA",1990,"USA"

Album(name, artist, year)

"Enter the Wu-Tang","Wu-Tang Clan",1993

"St.Ides Mix Tape","Wu-Tang Clan",1994

"Liquid Swords", "GZA", 1990

FLAT FILE STRAWMAN

Example: Get the year that GZA went solo.

Artist(name, year, country)

"Wu-Tang Clan",1992,"USA"

"Notorious BIG",1992,"USA"

"GZA",1990,"USA"



for line in file.readlines():
 record = parse(line)
 if record[0] == "GZA":
 print(int(record[1]))

FLAT FILES: DATA INTEGRITY

How do we ensure that the artist is the same for each album entry?

What if somebody overwrites the album year with an invalid string?

What if there are multiple artists on an album?

What happens if we delete an artist that has albums?

FLAT FILES: IMPLEMENTATION

How do you find a particular record?

What if we now want to create a new application that uses the same database?

What if two threads try to write to the same file at the same time?

FLAT FILES: DURABILITY

What if the machine crashes while our program is updating a record?

What if we want to replicate the database on multiple machines for high availability?

DATABASE MANAGEMENT SYSTEM

A <u>database management system</u> (**DBMS**) is software that allows applications to store and analyze information in a database.

A general-purpose DBMS supports the definition, creation, querying, update, and administration of databases in accordance with some <u>data model</u>.

A <u>data model</u> is a collection of concepts for describing the data in a database.

A <u>schema</u> is a description of a particular collection of data, using a given data model.

Relational

← Most DBMSs

Key/Value

Graph

Document / Object

Wide-Column / Column-family

Array / Matrix / Vectors

Hierarchical

Network

Multi-Value

Relational

Key/Value

Graph

Document / Object

Wide-Column / Column-family

Array / Matrix / Vectors

Hierarchical

Network

Multi-Value

← NoSQL

Relational

Key/Value

Graph

Document / Object

Wide-Column / Column-family

Array / Matrix / Vectors ← Machine Learning

Hierarchical

Network

Multi-Value

Relational

Key/Value

Graph

Document / Object

Wide-Column / Column-family

Array / Matrix / Vectors

Hierarchical

Network

Multi-Value

← Obsolete / Legacy / Rare

Relational

← This Course

Key/Value

Graph

Document / Object

Wide-Column / Column-family

Array / Matrix / Vectors

Hierarchical

Network

Multi-Value

EARLY DBMSs

Early database applications were difficult to build and maintain on available DBMSs in the 1960s.

- → Examples: <u>IDS</u>, <u>IMS</u>, <u>CODASYL</u>
- → Computers were expensive, humans were cheap.

Tight coupling between logical and physical layers.

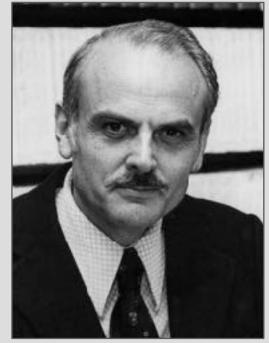
Programmers had to (roughly) know what queries the application would execute before they could deploy the database.

EARLY DBMSs

Ted Codd was a mathematician working at IBM Research in the late 1960s.

He saw IBM's developers spending their time rewriting database programs every time the database's schema or layout changed.

Devised the relational model in 1969.



Edgar F. Codd

DESIGNABILITY, RECOMMENCY AND CONSESSED OF RELATIONS. STORED IN LAKE DATA BARES

> E. T. Cald Despute the later Son Root, California

APPRACE: The large, integrated data banks of the fature will current many melations and about a degree is chared form. It will not be unusual for this set of stored relations to be nederlest. Two types of accordancy are detired and dispussed. One type may be employed to improve accommobility of pertain ideas of defaraction which repose to be in great desired. When cither two of peacedarty exists, these responsible for control of the dris basic should once about it and have some moans of detecting any "ingital" intersistencies in the total put of stocks selections. Combiners checking night be helpful in tracking dear unsubsyclicis (and possibly traudalent charges in the data bank porteril.

23 20004 (Glas) August 30, 1989

LEVING BENEFICIEN SCHOOL - This papers has been submitted for Mailingtion eloparate and hor taken deleted as a basement beyond that early Showbarles of its coleds. As a content to the latered publisher, in should not be whichly distributed world after the care of particles

Cooles are be investined than 189 Money In Notice Research Control Post SPECT for tall, includes beington, the York 2018.

Information Retrieval

late

iding

ase's

1969

P. MARNEAUE, Febru

A Relational Model of Data for Large Shared Data Banks

R. P. Cres. 1814 Bouard Leiszeiser, San Jose, California

Puters over of large data books need be profited from having to linew how the date is proprieted in the negligible line. interest representation). A prompting remice which supplies such information is act a satisfactory solution, extinities of users of terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the ecounty representation are dienard. Charges in data representation will after benameded on a result of changes in query, update, and report halfic and actual growth in the types of stored information.

Existing resinferential, formatted state endows preside users with translatered film or slightly man general extends march of the darw. in Section 1, mediagrammed Press madels. are discussed. A model bound on a very relation, in mornal facts fire clark those retailions, and the covered of its animals. dere salve grape are introduced, in Legion 2, contain appro-How on relations father than implest informace over charment and against to the problems of redundancy and considerary in the man's model.

are when any health, one part, one loss, not studies, date emperature, intermediate of white, emberging of white, exhibiting starbellishing reducting continues companies the mineral language, profession solubs, receiv, data temples

CREATIONES SITE SITE SITE 612, 612, 612

1. Relational Model and Normal Form

1.1. Dermoncornor

This paper is necessarily with the application of eleinjuriesy relation through to systems added provide chared among to large harlor of fore attend data. Energyl for a paper by Chible (t), the principal application of relations to date systems has been to deductive questions arresting systems. Levels and Micros. Mi provide numerous references to work in Distance.

In contrart, the problems treated here are those of data independence—the independence of application programs and terroinsi activities from growth in data types and changes in data representation—and seroids kinds of data increasing which are expected to become toroblesome over in mondadostive systems.

The relational view (or model) of data described in Section II appears to be expenied in several respects to the each or necessic model 12, 41 presently in segue for nonaferential systems. It provides a means of describing that with its natural structure unity—that it, without experies. point any additional elevature (as made in representation purposes. Amendingly, it provides a basis for a high less! date language white; will yield marked independence betwees programs on the one hand and marking recreasing tion and organization of data on the other.

A further solvenings of the relational view is that it. female second basis for treating derivability, redundancy. and constructed of relations—those are discussed in Section 2. The network model, on the other hand, has spreamed a number of senfedom, not the least of which is mistaking the desiration of connections for the deciration of relations see remarks in Section 2 on the "commerties trap").

Enally, the relational view permits a characteristics. of the stope and legical limitations of present formation. data systems, and also the relative movits (from a logisal standpoint) of competing separameters of data within a single system. Examples of this choice prospective are cited in various parts of this paper. Impresentations of systems to respect the relational model are not discussed.

1.2. Barra Dispusionario Princer Recessor

The precision of data description taking is mornely do. veloped information entires represents a major editator treated the grad of date independence [1, 6, 2]. Such tables buildate changing vertain characteristics of the date copys secution stored in a data basis. However, the variety of data representation characteristics which can be changed widout figurelly impairing some application programs to still gate limited. Further, the model of data with which mere intersect is still electrical with proposentational proportio, particularly in regard to the separatation of collections of data (as opposed to individual force). Three of the principal kinds of data dependencies which still need to be removed are: artisting dependence, including dependence, and secons path dependence. In some systems these dependencies are not clearly separable from one another.

121. Ordering Dependence, Elimente of dela in a data bank may be stored in a variety of ways, some involving no economy for ordering some permitting such element to participate in our ordering only, others possitting each element to participate in second inderings. Let us remailer these solding systems which either require as permit data. adjunctive to be expeed in at least one total codesing which in rively associated with the hardware determined ordering al addresses. Far reample, the remade of a Six concerning racts might be stored in ascending order by part serial number. Such systems normally permit application progenerate assume that the order of presentation of records from mach a file is identical to (or is a subcadering of) the

Volume 13 / Number 6 / June, 1809

Communications of the Max 1877

Tε

RELATIONAL MODEL

The relational model defines a database abstraction based on relations to avoid maintenance overhead.

Key tenets:

- → Store database in simple data structures (relations).
- → Physical storage left up to the DBMS implementation.
- → Access data through high-level language, DBMS figures out best execution strategy.

RELATIONAL MODEL

Structure: The definition of the database's relations and their contents.

Integrity: Ensure the database's contents satisfy constraints.

Manipulation: Programming interface for accessing and modifying a database's contents.

RELATIONAL MODEL

A <u>relation</u> is an unordered set that contain the relationship of attributes that represent entities.

A <u>tuple</u> is a set of attribute values (also known as its <u>domain</u>) in the relation.

- → Values are (normally) atomic/scalar.
- → The special value NULL is a member of every domain (if allowed).

Artist(name, year, country)

name	year	country
Wu-Tang Clan	1992	USA
Notorious BIG	1992	USA
GZA	1990	USA

n-ary Relation

Table with *n* columns

RELATIONAL MODEL: PRIMARY KEYS

A relation's <u>primary key</u> uniquely identifies a single tuple.

Some DBMSs automatically create an internal primary key if a table does not define one.

Auto-generation of unique integer primary keys:

- → **SEQUENCE** (SQL:2003)
- → AUTO_INCREMENT (MySQL)

Artist(id, name, year, country)

id	name	year	country
123	Wu-Tang Clan	1992	USA
456	Notorious BIG	1992	USA
789	GZA	1990	USA

RELATIONAL MODEL: FOREIGN KEYS

A <u>foreign key</u> specifies that an attribute from one relation has to map to a tuple in another relation.

RELATIONAL MODEL: FOREIGN KEYS

Artist(id, name, year, country)

id	name	year	country
123	Wu-Tang Clan	1992	USA
456	Notorious BIG	1992	USA
789	GZA	1990	USA

Album(id, name, artists, year)

id	name	artists	year
11	Enter the Wu-Tang	123	1993
22	St.Ides Mix Tape	???	1994
33	<u>Liquid Swords</u>	789	1995

RELATIONAL MODEL: FOREIGN KEYS



artist_id	album_id
123	11
123	22
789	22
456	22

Artist(id, name, year, country)

id	name	year	country
123	Wu-Tang Clan	1992	USA
456	Notorious BIG	1992	USA
789	GZA	1990	USA

Album(<u>id</u>, name, year)

id	name	year
11	Enter the Wu-Tang	1993
22	St.Ides Mix Tape	1994
33	<u>Liquid Swords</u>	1995

DATA MANIPULATION LANGUAGES (DML)

Methods to store and retrieve information from a database.

Procedural:

→ The query specifies the (high-level) strategy to find the desired result based on sets / bags. ← Relational Algebra

Non-Procedural (Declarative):

→ The query specifies only what data is wanted and not how to find it. ← Relational Calculus

RELATIONAL MODEL: QUERIES

The relational model is independent of any query language implementation.

SQL is the *de facto* standard (many dialects).

```
for line in file.readlines():
  record = parse(line)
  if record[0] == "GZA":
    print(int(record[1]))
```

SELECT year FROM artists WHERE name = 'GZA'

SQL HISTORY

In 1971, IBM created its first relational query language called <u>SQUARE</u>.

IBM then created "SEQUEL" in 1972 for <u>IBM</u>
<u>System R</u> prototype DBMS.

→ <u>Structured English Query Language</u>

IBM releases commercial SQL-based DBMSs:

→ System/38 (1979), SQL/DS (1981), and DB2 (1983).

SQI

In 1971, IBM created language called **SQ**

IBM then created "S System R prototype

→ Structured English C

Q2. Find the average salary of employees in the Shoc Department.

Mappings may be composed by applying one mapping to the result of another, as illustrated by Q3.

Q3. Find those items sold by departments on the second floor.

The floor '2' is first mapped to the departments located there, and then to the items which they sell. The range of the inner mapping must be compatible with the domain of the outer mapping, but they need not be identical, as illustrated by Q4.

IBM releases comme

→ System/38 (1979), SQL/DS (1981), and DB2 (1983).

SQL HISTORY

ANSI Standard in 1986. ISO in 1987

→ <u>Structured Query Language</u>

Current standard is SQL:2016

- → **SQL:2016** → JSON, Polymorphic tables
- → **SQL:2011** → Temporal DBs, Pipelined DML
- → **SQL:2008** → Truncation, Fancy Sorting
- → **SQL:2003** → XML, Windows, Sequences, Auto-Gen IDs.
- → **SQL:1999** → Regex, Triggers, OO

The minimum language syntax a system needs to say that it supports SQL is <u>SQL-92</u>.

SQL HIS

ANSI Standard in 1986. ISO

→ <u>Structured Query Language</u>

Current standard is SQL:201

- → **SQL:2016** → JSON, Polymorp
- → **SQL:2011** → Temporal DBs, P
- → **SQL:2008** → Truncation, Fanc
- → **SQL:2003** → XML, Windows,
- → **SQL:1999** → Regex, Triggers,

The minimum language syn say that it supports SQL is SQL-92.



RELATIONAL LANGUAGES

Data Manipulation Language (DML)
Data Definition Language (DDL)
Data Control Language (DCL)

Also includes:

- → View definition
- → Integrity & Referential Constraints
- → Transactions

Important: SQL is based on **bags** (duplicates) not **sets** (no duplicates).

RELATIONAL ALGEBRA

Fundamental operations to retrieve and manipulate tuples in a relation.

→ Based on set algebra.

Each operator takes one or more relations as its inputs and outputs a new relation.

→ We can "chain" operators together to create more complex operations.

σ Select

π Projection

Union

Intersection

Difference

× Product

× Join

RELATIONAL ALGEBRA: SELECT

Choose a subset of the tuples from a relation that satisfies a selection predicate.

- → Predicate acts as a filter to retain only tuples that fulfill its qualifying requirement.
- → Can combine multiple predicates using conjunctions / disjunctions.

Syntax: $\sigma_{\text{predicate}}(R)$

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a2	103
а3	104

 $\sigma_{a_id='a2'}(R)$

a_id	b_id
a2	102
a2	103

 $\sigma_{a_id='a2' \land b_id>102}(R)$

a_id	b_id
a2	103

SELECT * FROM R
WHERE a_id='a2' AND b_id>102

RELATIONAL ALGEBRA: PROJECTION

Generate a relation with tuples that contains only the specified attributes.

- → Can rearrange attributes' ordering.
- → Can manipulate the values.

Syntax: $\pi_{A1,A2,...,An}(R)$

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
a2	103
а3	104

 $\Pi_{\text{b_id-100,a_id}}(\sigma_{\text{a_id='a2'}}(R))$

b_id-100	a_id
2	a2
3	a2

SELECT b_id-100, a_id FROM R WHERE a_id = 'a2'

RELATIONAL ALGEBRA: UNION

Generate a relation that contains all tuples that appear in either only one or both input relations.

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
а3	103

S(a_id,b_id)

a_id	b_id
а3	103
a4	104
а5	105

Syntax: (R ∪ S)

(SELECT * FROM R)

UNION ALL

(SELECT * FROM S)

(R ∪ S)

a_id	b_id
a1	101
a2	102
а3	103
а3	103
a4	104
а5	105

RELATIONAL ALGEBRA: INTERSECTION

Generate a relation that contains only the tuples that appear in both of the input relations.

R(a_id,b_id)
a_id b_id

a_id	b_id
a1	101
a2	102
а3	103

S(a_id,b_id)

a_id	b_id
а3	103
a4	104
а5	105

Syntax: (R ∩ S)

(R ∩ S)

a_id	b_id
а3	103

(SELECT * FROM R)
INTERSECT
(SELECT * FROM S)

RELATIONAL ALGEBRA: DIFFERENCE

Generate a relation that contains only the tuples that appear in the first and not the second of the input relations.

Syntax: (R - S)

(SELECT * FROM R)

EXCEPT
(SELECT * FROM S)

R(a_id,b_id)

a_id	b_id
a1	101
a2	102
а3	103

S(a_id,b_id)

a_id	b_id
а3	103
a4	104
а5	105

(R - S)

a_id	b_id
a1	101
a2	102

RELATIONAL ALGEBRA: PRODUCT

Generate a relation that contains all possible combinations of tuples from the input relations.

Syntax: $(R \times S)$

SELECT * FROM R CROSS JOIN S

SELECT * FROM R, S

R(a_id,b_id)

1/01_1	<u></u>
a_id	b_id
a1	101
a2	102
а3	103

S(a_id,b_id)

<u> </u>	<u>,, — </u>
a_id	b_id
a3	103
a4	104
a5	105

 $(R \times S)$

R.a_id	R.b_id	S.a_id	S.b_id
a1	101	a3	103
a1	101	a4	104
a1	101	a5	105
a2	102	a3	103
a2	102	a4	104
a2	102	а5	105
a3	103	a3	103
a3	103	a4	104
a3	103	a5	105

RELATIONAL ALGEBRA: JOIN

R.a_id R.b_

103

a3

Generate a relation that contains all tuples that are a combination of two tuples (one from each input relation) with a common value(s) for one or more attributes.

Syntax: $(R \bowtie S)$

R(a_id,b_id))
--------------	---

a_id	b_id
a1	101
a2	102
а3	103

a_id S.b_id

103

S(a_id,b_id)

a_id	b_id
a3	103
a4	104
а5	105

(R ⋈ S)

a_id	b_id
а3	103

RELATIONAL ALGEBRA: JOIN

Generate a relation that contains all tuples that are a combination of two tuples (one from each input relation) with a common value(s) for one or more attributes.

Syntax: $(R \bowtie S)$

D		i a	h	i _A \
וח	(a_	ıu,	N_	_iu)

a_id	b_id
a1	101
a2	102
а3	103

S(a id,b id)

a_id	b_id
a3	103
a4	104
a5	105

(R ⋈ S)

a_id	b_id
а3	103

SELECT * FROM R NATURAL JOIN S

SELECT * FROM R JOIN S USING (a_id, b_id)

RELATIONAL ALGEBRA: EXTRA OPERATORS

```
Rename (ρ)
Assignment (R←S)
Duplicate Elimination (δ)
Aggregation (γ)
Sorting (τ)
Division (R÷S)
```

EXAMPLE DATABASE

student(sid,name,login,gpa)

sid	name	login	age	gpa
53666	Kanye	kanye@cs	45	4.0
53688	Bieber	jbieber@cs	29	3.9
53655	Tupac	shakur@cs	25	3.5

course(cid,name)

cid	name
15-445	Database Systems
15-721	Advanced Database Systems
15-826	Data Mining
15-799	Special Topics in Databases

enrolled(sid,cid,grade)

sid	cid	grade
53666	15-445	С
53688	15-721	Α
53688	15-826	В
53655	15-445	В
53666	15-721	С

Functions that return a single value from a bag of tuples:

- → AVG(col) → Return the average col value.
- → MIN(col) → Return minimum col value.
- → MAX(col) → Return maximum col value.
- → **SUM(col)** → Return sum of values in col.
- → COUNT(col) → Return # of values for col.

Aggregate functions can (almost) only be used in the **SELECT** output list.

Get # of students with a "@cs" login:

SELECT COUNT(login) AS cnt FROM student WHERE login LIKE '%@cs'

Aggregate functions can (almost) only be used in the **SELECT** output list.

Get # of students with a "@cs" login:

SELECT COUNT(login) AS cnt

SELECT COUNT(*) AS cnt
FROM student WHERE login LIKE '%@cs'

Aggregate functions can (almost) only be used in the **SELECT** output list.

Get # of students with a "@cs" login:

```
SELECT COUNT(login) AS cnt

SELECT COUNT(*) AS cnt

SELECT COUNT(1) AS cnt
FROM student WHERE login LIKE '%@cs'
```

MULTIPLE AGGREGATES

Get the number of students and their average GPA that have a "@cs" login.

	AVG(gpa)	COUN	T(sid)
SELECT AVG(gpa), COUNT(sid)	3.8	3	
FROM student WHERE login LIKE '%@cs'			

DISTINCT AGGREGATES

COUNT, SUM, AVG support DISTINCT

Get the number of unique students that have an "@cs" login.

SELECT COUNT(DISTINCT login)

FROM student WHERE login LIKE '%@cs'

Output of other columns outside of an aggregate is undefined.

Get the average GPA of students enrolled in each course.

	Ψ,		AVG(s.gpa)	e.cid
SELECT AVG(s.gpa),	⊁ ٍ ﴿	d	3.86	???
FROM enrolled AS	JL	N studer	nt AS s	
ON e.sid = s.sid				

GROUP BY

Project tuples into subsets and calculate aggregates against each subset.

SELECT AVG(s.gpa), e.cid FROM enrolled AS e JOIN student AS s ON e.sid = s.sid GROUP BY e.cid

e.sid	s.sid	s.gpa	e.cid
53435	53435	2.25	15-721
53439	53439	2.70	15-721
56023	56023	2.75	15-826
59439	59439	3.90	15-826
53961	53961	3.50	15-826
58345	58345	1.89	15-445



GROUP BY

Non-aggregated values in **SELECT** output clause must appear in **GROUP BY** clause.

SELECT AVG(s.gpa), e.cid, s.r. me
FROM enrolled AS e, student AS s
WHERE e.sid = s.sid
GROUP BY e.cid

GROUP BY

Non-aggregated values in **SELECT** output clause must appear in **GROUP BY** clause.

SELECT AVG(s.gpa), e.cid, s.name FROM enrolled AS e JOIN student AS s ON e.sid = s.sid GROUP BY e.cid, s.name

HAVING

Filters results based on aggregation, like a WHERE clause for a GROUP BY

SELECT AVG(s.gpa) AS avg_gpa, e.cid FROM enrolled AS e, student AS s

WHERE e.sid = s.sid

AND avg_gpa > 3.9

GROUP BY e.cid



HAVING

Filters results based on aggregation, like a WHERE clause for a GROUP BY

SELECT AVG(s.gpa) AS avg_gpa, e.cid FROM enrolled AS e, student AS s WHERE e.sid = s.sid GROUP BY e.cid HAVING avg_gpa > 3.9;

HAVING

Filters results based on aggregation, like a WHERE clause for a GROUP BY

SELECT AVG(s.gpa) AS avg_gpa, e.cid FROM enrolled AS e, student AS s WHERE e.sid = s.sid GROUP BY e.cid

HAVING AVG(s.gpa) > 3.9;

AVG(s.gpa)	e.cid
3.75	15-415
3.950000	15-721
3.900000	15-826



avg_gpa	e.cid
3.950000	15-721

ORDER BY <column*> [ASC | DESC]

→ Order the output tuples by the values in one or more of their columns.

OFLECT and awards FDOM arrestlant	SIC	grade
SELECT sid, grade FROM enrolled	53123	Α
WHERE cid = '15-721'	53334	Α
ORDER BY grade	53650	В
	53666	D

ORDER BY <column*> [ASC | DESC]

→ Order the output tuples by the values in one or more of their columns.

SELECT sid. grade FROM enrolled

W SELECT sid, grade FROM enrolled

WHERE cid = '15-721'

ORDER BY 1

SELECT sid FROM enrolled

WHERE cid = '15-721'

ORDER BY grade DESC, sid ASC

53650

53123

53334

ORDER BY <column*> [ASC | DESC]

→ Order the output tuples by the values in one or more of their columns.

SELECT sid. grade FROM enrolled

W SELECT sid, grade FROM enrolled

O WHERE cid = '15-721'

ORDER BY 1

SELECT sid FROM enrolled

W SELECT sid FROM enrolled

O WHERE cid = '15-721'

ORDER BY grade DESC, 1 ASC

LIMIT <count> [offset]

- → Limit the # of tuples returned in output.
- → Can set an offset to return a "range"

SELECT sid, name FROM student WHERE login LIKE '%@cs' LIMIT 10

SELECT sid, name FROM student WHERE login LIKE '%@cs' LIMIT 20 OFFSET 10

Queries containing other queries. They are often difficult to optimize.

Inner queries can appear (almost) anywhere in the query.

Outer Query
SELECT name FROM student
WHERE
sid IN (SELECT sid FROM enrolled)
Inner Query

ALL→ Must satisfy expression for all rows in the sub-query.

ANY→ Must satisfy expression for at least one row in the sub-query.

IN→ Equivalent to '=ANY()'.

EXISTS→ At least one row is returned without comparing it to an attribute in outer query.

Get the names of students in '15-445'

```
SELECT name FROM student
WHERE sid IN (
SELECT sid FROM enrolled
WHERE cid = '15-445'
)
```

Get the names of students in '15-445'

```
SELECT name FROM student

W SELECT name FROM student

WHERE sid = ANY (
SELECT sid FROM enrolled
WHERE cid = '15-445'
)
```

Find all courses that have no students enrolled in them.

SELECT * FROM course WHERE ...

"with no tuples in the enrolled table"

cid	name
15-445	Database Systems
15-721	Advanced Database Systems
15-826	Data Mining
15-799	Special Topics in Databases

sid	cid	grade
53666	15-445	С
53688	15-721	Α
53688	15-826	В
53655	15-445	В
53666	15-721	С

Find all courses that have no students enrolled in them.

```
SELECT * FROM course
WHERE NOT EXISTS (
tuples in the enrolled table
)
```

Find all courses that have no students enrolled in them.

```
SELECT * FROM course
WHERE NOT EXISTS (
SELECT * FROM enrolled
WHERE course.cid = enrolled.cid
)
```

cid	name
15-799	Special Topics in Databases

CONCLUSION

Databases are ubiquitous.

Relational algebra defines the primitives for processing queries on a relational database.

We will see relational algebra again when we talk about query optimization + execution.

SQL is not a dead language.

NEXT CLASS

Disk Storage