Databases 1

Daniel POP

Week 2 - 4

Agenda

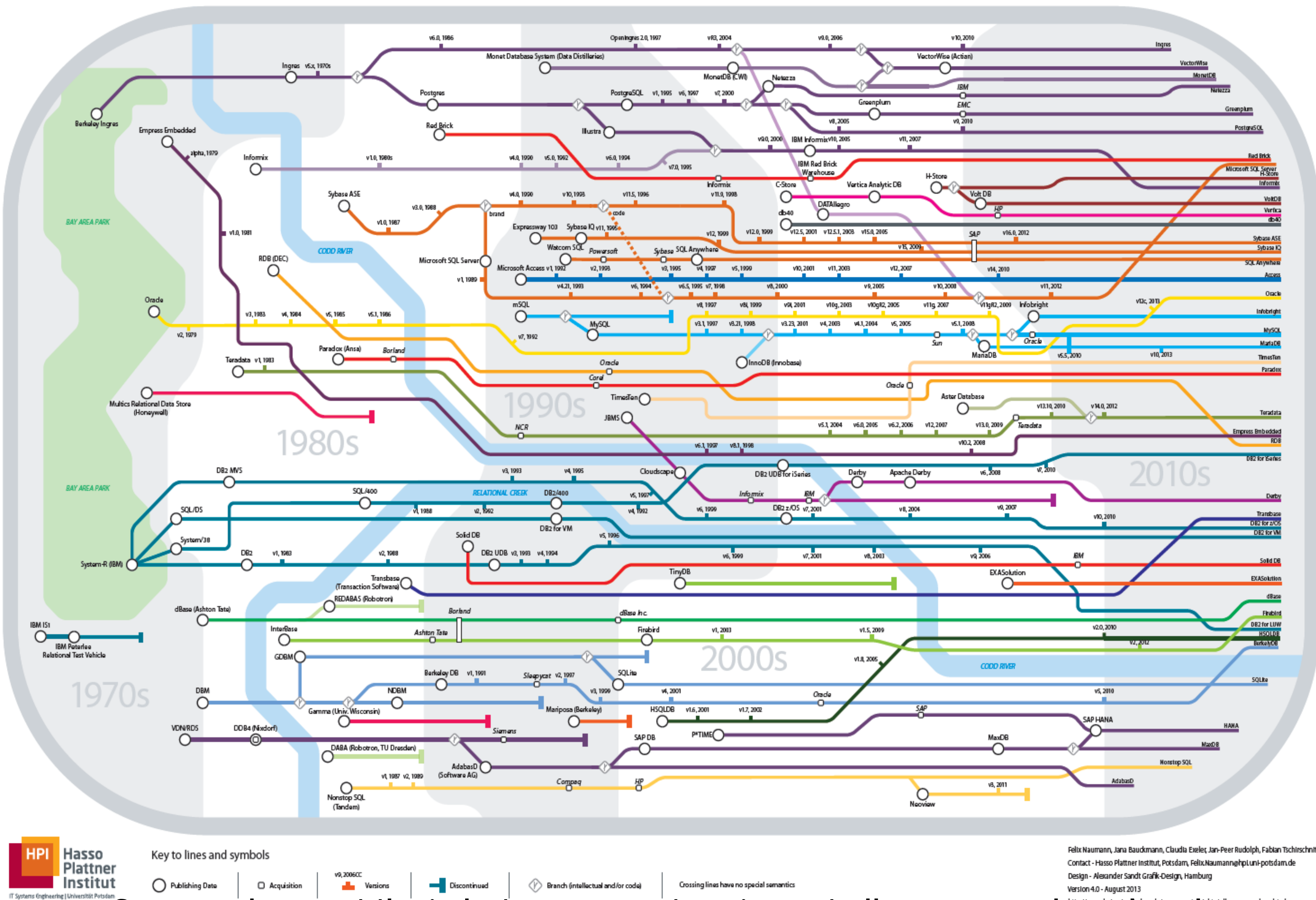
The Relational Model

1. Origins and history
2. Key concepts
3. Relational integrity
4. Relational algebra
5. SQL implementation
6. 12+1 Codd rules for a relational DBMS

Brief history

- Proposed by E.F. Codd in 1970 (A relational model of data for large shared data banks)
  - High degree of data independence
  - Dealing with data semantics, consistency, and redundancy
  - Introduces the concept of normalization
- System R, developed by IBM at San Jose Research Laboratory California, in the late 1970s
  - Led to the development of SQL
  - Initiated the production of commercial RDBMSs
- INGRES (Interactive Graphics REtrieval System) at they University of California at Berkley

Genealogy of Relational Database Management Systems



Source: https://hpi.de/naumann/projects/rdbms-genealogy.html

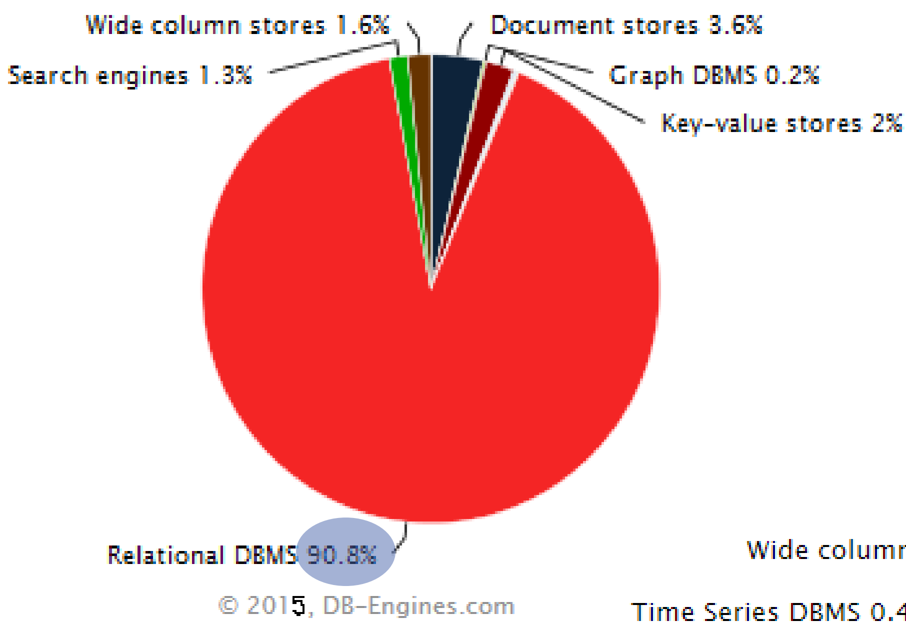
Most popular model, used by all major DBMS

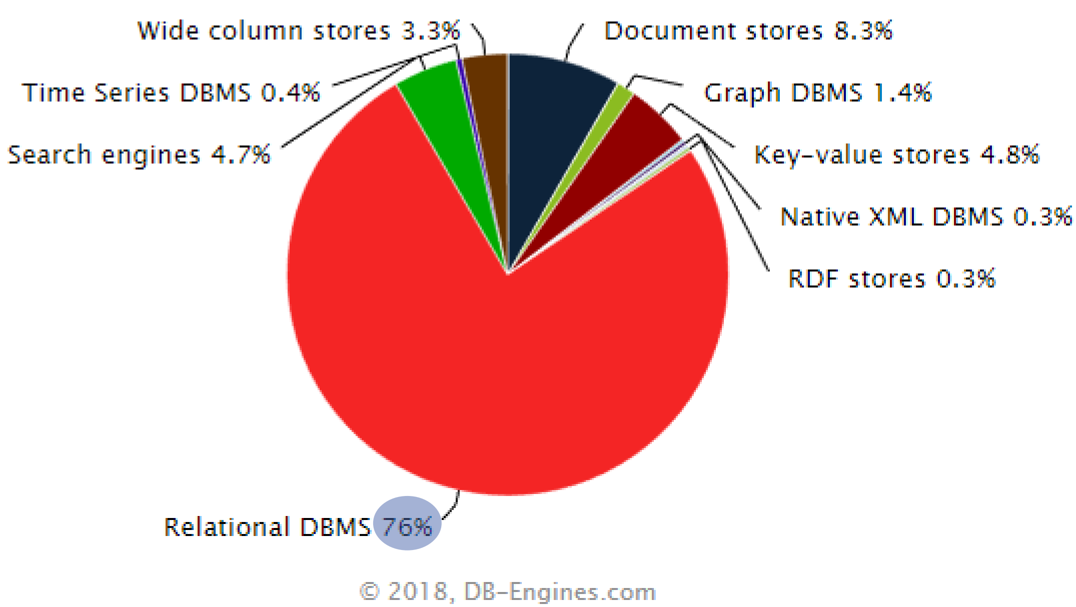
Rank Oct 2015	Rank Sep 2015	Rank Oct 2014	DBMS	Database Model	Score Oct 2015	Score Sep 2015	Score Oct 2014
1.	1.	1.	Oracle	Relational DBMS	1466.95	+3.58	-4.95
2.	2.	2.	MySQL	Relational DBMS	1278.96	+1.21	+15.99
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1123.23	+25.40	-96.37
4.	4.	↑ 5.	MongoDB	Document store	293.27	-7.30	+52.86
5.	5.	↓ 4.	PostgreSQL	Relational DBMS	282.13	-4.05	+24.41
6.	6.	6.	DB2	Relational DBMS	206.81	-2.33	-0.86
7.	7.	7.	Microsoft Access	Relational DBMS	141.83	-4.17	+0.19
8.	8.	↑ 10.	Cassandra	Wide column store	129.01	+1.41	+43.30
9.	9.	↓ 8.	SQLite	Relational DBMS	102.67	-4.99	+7.71
10.	10.	↑ 12.	Redis	Key-value store	98.80	-1.86	+19.42

Rank Oct 2018	Rank Sep 2018	Rank Oct 2017	DBMS	Database Model	Score Oct 2018	Score Sep 2018	Score Oct 2017
1.	1.	1.	Oracle	Relational DBMS	1319.27	+10.15	-29.54
2.	2.	2.	MySQL	Relational DBMS	1178.12	-2.36	-120.71
3.	3.	3.	Microsoft SQL Server	Relational DBMS	1058.33	+7.05	-151.99
4.	4.	4.	PostgreSQL	Relational DBMS	419.39	+12.97	+46.12
5.	5.	5.	MongoDB	Document store	363.19	+4.39	+33.79
6.	6.	6.	DB2	Relational DBMS	179.69	-1.38	-14.90
7.	↑ 8.	↑ 9.	Redis	Key-value store	145.29	+4.35	+23.24
8.	↓ 7.	↑ 10.	Elasticsearch	Search engine	142.33	-0.28	+22.09
9.	9.	↓ 7.	Microsoft Access	Relational DBMS	136.80	+3.41	+7.35
10.	10.	↓ 8.	Cassandra	Wide column store	123.39	+3.83	-1.40

Source: http://db-engines.com/en/ranking

Most popular model, used by all major DBMS





Key Characteristics

- Very simple model
- Ad-hoc query with high-level languages (SQL)
- Efficient implementations

Relational Model Concepts

- The relational model consists of the following:
  - Collection of relations
  - Set of operators to act on the relations
  - Data integrity for accuracy and consistency

- Intension (Schema) vs. Extension of a relational database
  - Schema is a structural description of all relations
  - Instance (extension) is the actual content a given point in time of the

Terminology

- Relational database = a collection of normalized relations
- Relation = a table with columns and rows
- Attribute = a named column of a relation
  - Domain = a set of allowable values for one or more attributes
  - SQL Data Types
- Tuple = a row of a relation
- Degree = the number of attributes contained in a relation
- Cardinality = the number of tuples of a relation

Relational Database Definition

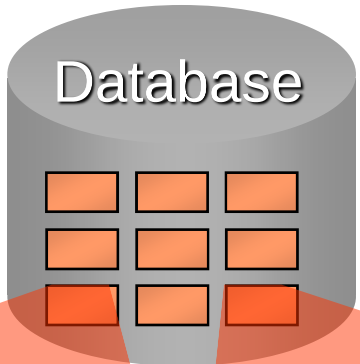


Table Name: EMP

EMPNO	ENAME	JOB	DEPTNO
7839	KING	PRESIDENT	10
7698	BLAKE	MANAGER	30
7782	CLARK	MANAGER	10
7566	JONES	MANAGER	20

Table Name: DEPT

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Database relations

- Relation schema is a relation name followed by a set {Ai:Di} of attribute and domain name pairs

$$R = \{A_1{:}D_1,\ A_2{:}D_2,\ \dots\ ,\ A_n{:}D_n\}$$

- Properties of relations:
  - The name is unique
  - Each cell contains exactly one atomic value
  - Attribute names are distinct
  - The values of an attribute are all from the same domain
  - The order of attributes has no significance
  - The order of tuples has no significanct

Running Example

- A relational database for student enrollment:

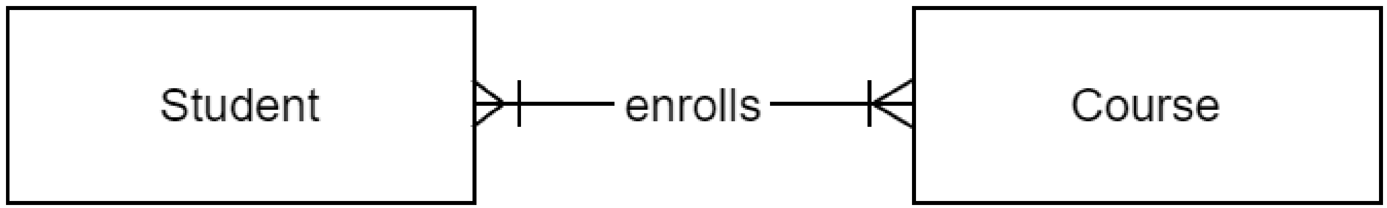
  Students of our university should enroll in courses they want to attend. One student may enroll in up to 8 courses and for a course to be run it is required at least 10 accepted students. As places in courses are limited, for each enrollment request there will a decision associated whether the student is accepted or not in the course he/she opted for.

  Courses are offered by different departments of our university, are uniquely identified by their title and each course is credited a fixed number of credits for all students enrolled into.

  Students may enroll to courses offered by different departments.

Example

- A relational database for student enrollment:



Example

- A relational database for student enrollment:

```
Courses(CourseTitle:NVARCHAR(50), Department:NVARCHAR(20),
Credits:INTEGER)

Students(StudID:INTEGER, StudName:NVARCHAR(50), DoB:DATE,
PoB:NVARCHAR(50), Major:NVARCHAR(40))

Enrollments(StudID:INTEGER, CourseTitle:NVARCHAR(50),
EnrollmentDate:DATE, Decision:BOOLEAN)
```

Relational Keys

- Superkey = an attribute or set of attributes that uniquely identifies a tuple within a relation

- Keys can be
  - Single attribute = a key consisting of exactly one attribute
  - Composite key = a key consisting of more than one attribute

- Candidate key = a superkey such that no proper subset is a superkey within the relation
  - Uniqueness – the values of the candidate key uniquely identify each tuple
  - Irreductibility – no proper subset of K has the uniqueness property

- Candidate key can be
  - Primary Key = a candidate key selected by the database designer to uniquely identify tuples within a relation
  - Alternate Keys = all other candidate keys, except the one elected to be the primary key

- Foreign key = an attribute or a set of attributes within one relation that matches the candidate key of other (possibly the same) relation

Exercise

- Identify the superkeys, candidate keys, primary keys and foreign keys in the previous example

Exercise

- Identify the superkeys, candidate keys, primary keys and foreign keys in the previous example

```
Courses(CourseTitle:CHAR(50), Department:CHAR(20),
Credits:INTEGER)

Students(StudID:INTEGER, StudName:CHAR(50), DoB:DATE,
PoB:CHAR(50), Major:CHAR(40))

Enrollments(StudID:INTEGER, CourseTitle:CHAR(50),
EnrollmentDate:DATE, Decision:BOOLEAN)
```

SQL Implementation of relation model

- Relations are mapped to SQL tables

```
CREATE TABLE Students (
   StudID int NOT NULL,
   StudName varchar(50),
   DoB date,
   PoB varchar(50),
   Major varchar(40));
```

`ALTER TABLE` – change table's schema: add/remove columns, add constraints etc.

SQL Implementation of relation model

- Setting up Primary Key in different ways:
  - While creating the table for single-attribute primary keys

```
CREATE TABLE Students (
     StudID int PRIMARY KEY,
...);
```

  - While creating the table for composed primary keys

```
CREATE TABLE Students (
     ...,
     CONSTRAINT PK_Students PRIMARY KEY (StudentName, DoB, PoB) );
```

  - Later on by modifying table's structure:

```
ALTER TABLE Students
ADD PRIMARY KEY (StudentID)

ALTER TABLE Students
ADD CONSTRAINT PK_Students PRIMARY KEY (StudentName, DoB, PoB)
```

- Removing Primary Key

```
ALTER TABLE Students
DROP PRIMARY KEY

ALTER TABLE Students
DROP CONSTRAINT PK_Students
```

SQL Implementation of relation model

- Setting up Alternate Key (Unique constraint) in different ways:
  - While creating the table for single-attributed unique constraint

```
CREATE TABLE Students (
    SomeColumn int NOT NULL UNIQUE,
...);
```

  - While creating the table for composed unique constraints

```
CREATE TABLE Students (
    ...,
    CONSTRAINT AK_Students_StudName_DoB_PoB
    UNIQUE (StudentName, DoB, PoB) );
```

  - Later on by modifying table's structure:

```
ALTER TABLE Students
ADD UNIQUE (SomeColumn)

ALTER TABLE Students
ADD CONSTRAINT AK_Students_StudName_DoB_PoB
   UNIQUE (StudentName, DoB, PoB)
```

- Removing Primary Key

```
ALTER TABLE Students
DROP CONSTRAINT AK_Students_StudName_DoB_PoB
```

SQL Implementation of relation model

- Setting up Foreign Key in different ways:
  - While creating the table for single-attribute foreign keys

```
CREATE TABLE Enrollments (
     StudID int FOREIGN KEY REFERENCES Students(StudID),
...)
```

  - While creating the table for composed foreign keys

```
CREATE TABLE Enrollments (
     ...,
     CONSTRAINT FK_Courses_Enrollments
         FOREIGN KEY (CourseTitle)
         REFERENCES Courses(CourseTitle))
```

  - Later on by modifying table's structure:

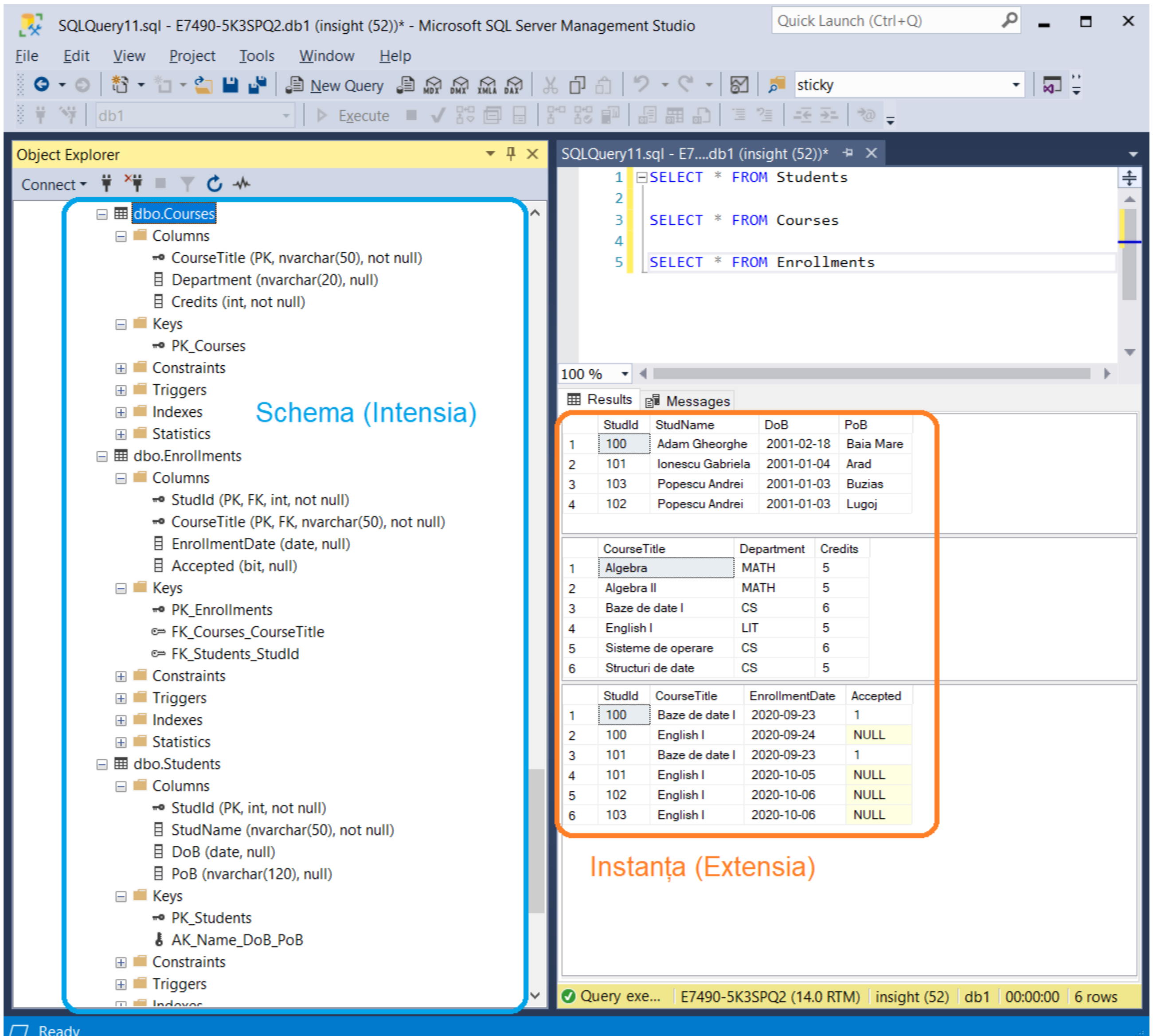
```
ALTER TABLE Enrollments
ADD FOREIGN KEY (StudID) REFERENCE Students(StudID)

ALTER TABLE Enrollments
ADD CONSTRAINT FK_Courses_Enrollments
         FOREIGN KEY (CourseTitle)
         REFERENCES Courses(CourseTitle)
```

- Removing Foreign Key

```
ALTER TABLE Enrollments
DROP CONSTRAINT FK_Courses_Enrollments
```

Exercise


SQLQuery11.sql - E7490-5K3SPQ2.db1 (insight (52))* - Microsoft SQL Server Management Studio
Quick Launch (Ctrl+Q)
File Edit View Project Tools Window Help
New Query
sticky
db1
Execute
Object Explorer
Connect
dbo.Courses
Columns
CourseTitle (PK, nvarchar(50), not null)
Department (nvarchar(20), null)
Credits (int, not null)
Keys
PK_Courses
Constraints
Triggers
Indexes
Statistics
dbo.Enrollments
Columns
StudId (PK, FK, int, not null)
CourseTitle (PK, FK, nvarchar(50), not null)
EnrollmentDate (date, null)
Accepted (bit, null)
Keys
PK_Enrollments
FK_Courses_CourseTitle
FK_Students_StudId
Constraints
Triggers
Indexes
Statistics
dbo.Students
Columns
StudId (PK, int, not null)
StudName (nvarchar(50), not null)
DoB (date, null)
PoB (nvarchar(120), null)
Keys
PK_Students
AK_Name_DoB_PoB
Constraints
Triggers
Schema (Intensia)
SQLQuery11.sql - E7....db1 (insight (52))*
SELECT * FROM Students
SELECT * FROM Courses
SELECT * FROM Enrollments
100 %
Results
Messages
StudId StudName DoB PoB
1 100 Adam Gheorghe 2001-02-18 Baia Mare
2 101 Ionescu Gabriela 2001-01-04 Arad
3 103 Popescu Andrei 2001-01-03 Buzias
4 102 Popescu Andrei 2001-01-03 Lugoj
CourseTitle Department Credits
1 Algebra MATH 5
2 Algebra II MATH 5
3 Baze de date I CS 6
4 English I LIT 5
5 Sisteme de operare CS 6
6 Structuri de date CS 5
StudId CourseTitle EnrollmentDate Accepted
1 100 Baze de date I 2020-09-23 1
2 100 English I 2020-09-24 NULL
3 101 Baze de date I 2020-09-23 1
4 101 English I 2020-10-05 NULL
5 102 English I 2020-10-06 NULL
6 103 English I 2020-10-06 NULL
Instanța (Extensia)
Query exe... E7490-5K3SPQ2 (14.0 RTM) insight (52) db1 00:00:00 6 rows
Ready

Example

Branch	(branchNo, street, city, postcode)
Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent	(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)
Client	(clientNo, fName, lName, telNo, prefType, maxRent)
PrivateOwner	(ownerNo, fName, lName, address, telNo)
Viewing	(clientNo, propertyNo, viewDate, comment)
Registration	(clientNo, branchNo, staffNo, dateJoined)

Taken from *Database Systems - A Practical Approach to Design, Implementation, and Management (4th edition)* by Thomas Connolly and Carolyn Begg, Addison-Wesley, 2004 |

Exercise

- Given relation R(A, B, C, D) the following sets of attributes uniquely identifies the tuples in the relation {A}, {B, C}. Fill in the following table

Set	Superkey	Candidate key	Composite key	Foreign key
{A}				
{A, B}				
{B, C}				
{A, C}				
{B}				
{B, D}				

Exercise

- Given relation R(A, B, C, D) the following sets of attributes uniquely identifies the tuples in the relation {A}, {B, C}. Fill in the following table

Set	Superkey	Candidate key	Composite key	Foreign key
{A}	**YES**	**YES**	NO	?
{A, B}	**YES**	NO	**YES**	?
{B, C}	**YES**	**YES**	**YES**	?
{A, C}	**YES**	NO	**YES**	?
{B}	NO	NO	NO	?
{B, D}	NO	NO	NO	?

Relational Integrity

- Null = a value for an attribute that is currently unknown (undefined)
- Integrity rules: (see next slide)
- General constraints: additional rules specified by the data / database administrators that define or constrain some aspects of the enterprise.
- Domain constraints: constraints

Relational Integrity

- Entity Integrity: in a relation/table, a primary key of a tuple, or any part of it, can never take a null value.

- Referential Integrity: if a foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation or the foreign key value must be wholly null.

Querying relational model

- Relational Algebra
  - formal
- Structured Query Language (SQL)
  - de facto/implemented
- SQL is also used for DML and DDL
- Some queries easy to pose, some more difficult
- Some easy to execute, others more difficult (expensive)

Querying relational model: examples

- List name and date of birth of all students with major in CS – natural language (plain English)

- Relational Algebra – formal

  $\Pi_{\text{StudName, DoB}}(\sigma_{\text{Major='CS'}}\text{ (Students))}$

- Structured Query Language (SQL) – de facto/implemented

```
SELECT StudName, DoB
FROM Students
WHERE Major='CS'
```

Relational Algebra

Relational Algebra

- Theoretical language with operations that work on one or more relations
- Both the operands and the results are relations
- Closure = relations are closed under the algebra
- Operations (operators)
  - **Selection (filter)**
  - **Projection (slice)**
  - Join (combine)
  - Set-based operations
    - **Cartesian Product (cross-product)**
    - **Union**
    - **Set Difference**
    - Intersection
  - Rename
- Remark: duplicated tuples are purged from the result
- Remark: **bold** operators originally defined by **E.F Codd** in 1970

Table name

- R
  - The simplest query
  - Returns the copy of the relation
  - Examples:
    - Students
    - Enrollments

Selection

- $\sigma_{\text{predicate}}(R)$ $\quad \sigma_{P}(R) := \{t \mid t \in R \land P(t) = true\}$

CourseTitle	Department	Credits
Algebra	MATH	5
Algebra II	MATH	5
Baze de date I	CS	6
English I	LIT	5
Sisteme de operare	CS	6
Structuri de date	CS	5

- Works on a single relation R and returns the subset of relation R that contains only those tuples satisfying the specified condition (predicate)
- It is used to filter tuples of relation R based on a predicate

- Examples:
  - Students with Major in CS:
  - Students accepted in Databases course :

Selection

CourseTitle	Department	Credits
Algebra	MATH	5
Algebra II	MATH	5
Baze de date I	CS	6
English I	LIT	5
Sisteme de operare	CS	6
Structuri de date	CS	5

- $\sigma_{\text{predicate}}$ (R) $\quad \sigma_P(R) := \{t | t \in R \wedge P(t) = \mathit{true}\}$
  - Works on a single relation R and returns the subset of relation R that contains only those tuples satisfying the specified condition (predicate)

English	Students with major in CS
Relational Alg.	$\sigma_{\text{Major='CS'}}$ (Students)
SQL	SELECT * FROM Students WHERE Major='CS'

English	Students accepted in Databases course
Relational Alg.	$\sigma_{\text{CourseTitle='Databases'} \wedge \text{Decision=TRUE}}$(Enrollments)
SQL	SELECT * FROM Enrollments WHERE CourseTitle='Databases' AND Decision = TRUE

Projection

- $\Pi_{col1,\ldots,coln}(R)$ $\pi_\beta(R) := \{t_\beta | t \in R\}$

CourseTitle	Department	Credits
Algebra	MATH	5
Algebra II	MATH	5
Baze de date I	CS	6
English I	LIT	5
Sisteme de operare	CS	6
Structuri de date	CS	5

- Works on a single relation R and returns a new relation that contains a vertical subset of R, extracting the values of specified attributes and *eliminating duplicates*
- Examples:
  - Name and major of all students:

Projection

- $\Pi_{col1,\dots,coln}(R)$ $\pi_\beta(R) := \{t_\beta | t \in R\}$

CourseTitle	Department	Credits
Algebra	MATH	5
Algebra II	MATH	5
Baze de date I	CS	6
English I	LIT	5
Sisteme de operare	CS	6
Structuri de date	CS	5

  - Works on a single relation R and returns a new relation that contains a vertical subset of R, extracting the values of specified attributes and *eliminating duplicates*

English	Name and major of all students
Relational Alg.	$\Pi_{\text{StudName, Major}}$(Students)
SQL	SELECT [DISTINCT] StudName, Major FROM Students

English	CourseTitle and department
Relational Alg.	$\Pi_{\text{CourseTitle, Department}}$(Courses)
SQL	SELECT [DISTINCT] CourseTitle, Department FROM Courses

Projection

- Remark:
  - In Relational Algebra, duplicates are eliminated (set theory)
  - In SQL, duplicates are not;
    - There is a SELECT DISTINCT that does eliminate duplicates.

Assignment statements

- Complex queries may be broken down into simpler expressions

- Example

$$\Pi_{\text{StudName, DoB}}(\sigma_{\text{Major='CS'}}(\text{Students}))$$

is equivalent to

$$R1 := \sigma_{\text{Major='CS'}}(\text{Students})$$
$$R2 := \Pi_{\text{StudName, DoB}}(R1)$$

Cartesian/Cross- Product

A1(Name, Roll No)

Name	Roll No
Anoop	1
Anurag	2

X

A2(Name, Roll No)

Name	Roll No
Anoop	1
Anurag	2
Ganesh	3

Query Output(A1 X A2)

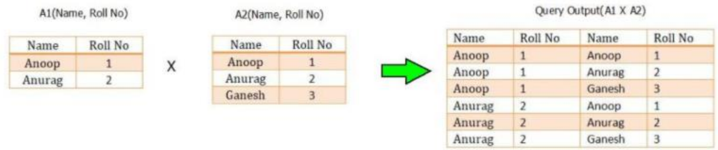
Name	Roll No	Name	Roll No
Anoop	1	Anoop	1
Anoop	1	Anurag	2
Anoop	1	Ganesh	3
Anurag	2	Anoop	1
Anurag	2	Anurag	2
Anurag	2	Ganesh	3

- R x S $R \times S := \{(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m) | (a_1, a_2, \ldots, a_n) \in R \wedge (b_1, b_2, \ldots, b_m) \in S\}$
  - Returns a new relation that is the concatenation of every tuple of relation R with each tuple of relation S

- The schema of the cross-product relation is the union of the 2 schemas of the operands relations

  - How many tuples in the Cartesian product of R x S?
    - **Cardinality(R) x Cardinality(S)**
    - **If cross-product two tables of 100 rows each it yields a table with 10,000 rows !!**

Cartesian/Cross-Product



A1(Name, Roll No)

Name	Roll No
Anoop	1
Anurag	2

x

A2(Name, Roll No)

Name	Roll No
Anoop	1
Anurag	2
Ganesh	3

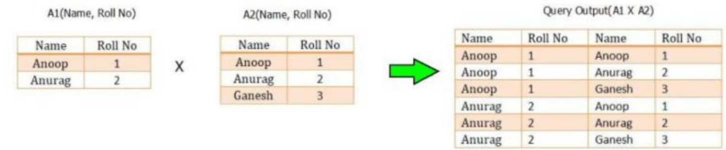
Query Output(A1 X A2)

Name	Roll No	Name	Roll No
Anoop	1	Anoop	1
Anoop	1	Anurag	2
Anoop	1	Ganesh	3
Anurag	2	Anoop	1
Anurag	2	Anurag	2
Anurag	2	Ganesh	3

- R x S $R \times S := \{(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m) | (a_1, a_2, \ldots, a_n) \in R \wedge (b_1, b_2, \ldots, b_m) \in S\}$
  - Returns a new relation that is the concatenation of every tuple of relation R with each tuple of relation S
  - Examples

English	All students times all enrollments
Relational Alg.	Students x Enrollments
SQL	SELECT * FROM Students, Enrollments, or SELECT * FROM Students CROSS JOIN Enrollments

Cartesian/Cross-Product



A1(Name, Roll No)

Name	Roll No
Anoop	1
Anurag	2

X

A2(Name, Roll No)

Name	Roll No
Anoop	1
Anurag	2
Ganesh	3

Query Output(A1 X A2)

Name	Roll No	Name	Roll No
Anoop	1	Anoop	1
Anoop	1	Anurag	2
Anoop	1	Ganesh	3
Anurag	2	Anoop	1
Anurag	2	Anurag	2
Anurag	2	Ganesh	3

- R x S $R \times S := \{(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m) | (a_1, a_2, \ldots, a_n) \in R \wedge (b_1, b_2, \ldots, b_m) \in S\}$
  - Returns a new relation that is the concatenation of every tuple of relation R with each tuple of relation S
  - Examples

English	All students and their enrollments
Relational Alg.	σ $_{\text{Students.StudID=Enrollments.StudID}}$ (Students x Enrollments))
SQL	SELECT * FROM Students, Enrollments WHERE Students.StudID = Enrollments.StudID

Cartesian/Cross-Product

A1(Name, Roll No)

Name	Roll No
Anoop	1
Anurag	2

X

A2(Name, Roll No)

Name	Roll No
Anoop	1
Anurag	2
Ganesh	3

Query Output(A1 X A2)

Name	Roll No	Name	Roll No
Anoop	1	Anoop	1
Anoop	1	Anurag	2
Anoop	1	Ganesh	3
Anurag	2	Anoop	1
Anurag	2	Anurag	2
Anurag	2	Ganesh	3

- R x S $R \times S := \{(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m) | (a_1, a_2, \ldots, a_n) \in R \wedge (b_1, b_2, \ldots, b_m) \in S\}$
  - Returns a new relation that is the concatenation of every tuple of relation R with each tuple of relation S
  - Examples

English	Return the name and major of all students accepted in 'English' course
Relational Alg.	$\Pi_{\text{StudName, Major}}(\sigma_{\text{Students.StudID=Enrollments.StudID} \wedge \text{CourseTitle='English'} \wedge \text{Decision=TRUE}}(\text{Students x Enrollments}))$
SQL	SELECT StudName, Major FROM Students CROSS JOIN Enrollemnts WHERE Students.StudID = Enrollments.StudID AND CourseTitle='English' AND Decision=TRUE

Rename

- $\rho_{R(A_1, \ldots, A_n)}(Exp)$

- Usages: Disambiguation in self-joins

Rename

- $\rho_{R(A_1, \ldots, A_n)}(Exp)$

- Usages: Disambiguation in self-joins

- Example:

English	Return pairs of courses offered by the same department
Relational Alg.	$\sigma_{D1=D2}(\rho_{C1(CT1, D1, C1)}(Courses) \times \rho_{C2(CT2, D2, C2)}(Courses))$
SQL	SELECT * FROM Courses AS C1, Courses AS C2 WHERE C1.Department = C2.Department

Rename

- $\rho_{R(A_1, \ldots, A_n)}(Exp)$

- Usages: Disambiguation in self-joins

- Example:

English	Return pairs of distinct courses offered by the same department
Relational Alg.	$\sigma_{D1=D2 \wedge CT1 <> CT2} (\rho_{C1(CT1,\ D1,\ C1)}(Courses) \times \rho_{C2(CT2,\ D2,\ C2)}(Courses))$
SQL	SELECT * FROM Courses AS C1, Courses AS C2 WHERE C1.Department = C2.Department AND C1.CourseTitle <> C2.CourseTitle

Exercise

Which of the following expressions does NOT return the name and major of students born in Timisoara who applied for Databases course and were rejected?

a) $\pi_{StudName,Major}(\sigma_{Students.StudID=Enrollments.StudID}(\sigma_{PoB='Timisoara'}(Students)\times\sigma_{CourseTitle='Databases' \wedge Decision=FALSE}(Enrollments)))$

b) $\pi_{StudName,Major}(\sigma_{Students.StudID=Enrollments.StudID \wedge PoB='Timisoara' \wedge CourseTitle='Databases' \wedge Decision=FALSE}(Students\times\pi_{StudentsID,CourseTitle,Decision}(Enrollments)))$

c) $\sigma_{Students.StudID=Enrollments.StudID}(\pi_{StudName,Major}(\sigma_{PoB='Timisoara'}(Students \times \sigma_{CourseTitle='Databases' \wedge Decision=FALSE}(Enrollments)))$

Exercise

Which of the following expressions does NOT return the name and major of students born in Timisoara who applied for Databases course and were rejected?

a) $\pi_{StudName,Major}(\sigma_{Students.StudID=Enrollments.StudID}(\sigma_{PoB='Timisoara'}(Students)\times\sigma_{CourseTitle='Databases' \wedge Decision=FALSE}(Enrollments)))$

b) $\pi_{StudName,Major}(\sigma_{Students.StudID=Enrollments.StudID \wedge PoB='Timisoara' \wedge CourseTitle='Databases' \wedge Decision=FALSE}(Students\times\pi_{StudentsID,CourseTitle,Decision}(Enrollments)))$

c) $\sigma_{Students.StudID=Enrollments.StudID}(\pi_{StudName,Major}(\sigma_{PoB='Timisoara'}(Students \times \sigma_{CourseTitle='Databases' \wedge Decision=FALSE}(Enrollments)))$

The correct answer is c) because after projection on StudName and Major the attributes StudID in Students/Enrollments are no longer available.

Join Operations

- Typically we only need subsets of the Cartesian product

- Types of joins:
  - Theta join
  - Equi join
  - Natural join

- Later renamed to INNER Join

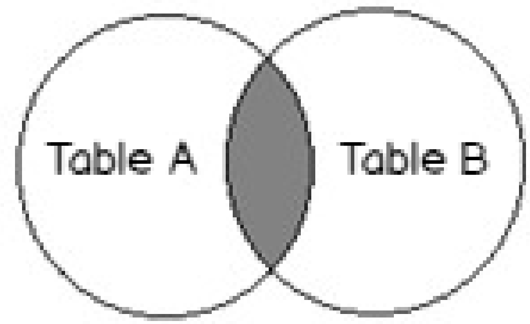
- No additional power to Relational Algebra; these are shortened forms of other expressions

- The schema of join operation is the union of the 2 schemas of the operands relations

Theta Inner Join

$R \bowtie_F S$

$R \bowtie_{A\theta B} S := \{r \cup s \mid r \in R \wedge s \in S \wedge r_{[A]} \ \theta \ s_{[B]}\}$



Returns a new relation that contains tuples satisfying the predicate F from the Cartesian product of R and S.

The predicate F is of the form

$$F = R.a_i \ \theta \ S.b_i$$

where θ may be one of the comparison operators:

$<, >, <=, >=, =, <>$

A Theta join is a shortened form of: $R \bowtie_F S = \sigma_F (R \times S)$

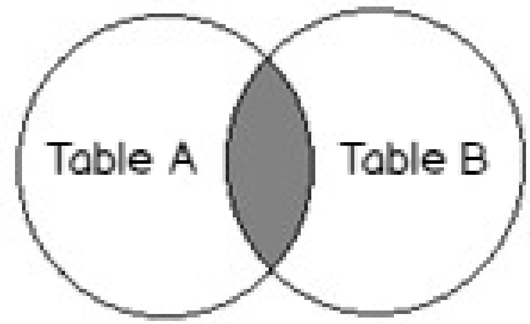
SELECT * FROM R, S WHERE F

SELECT * FROM R, S WHERE R.Ai [>|<|>=|<=|=] S.Bj

Equi Inner Join

$R \bowtie_F S$ where F is like $R.a_i = S.b_i$

$$R \bowtie_{A=B} S := \{r \cup s \mid r \in R \wedge s \in S \wedge r_{[A]} = s_{[B]}\}$$



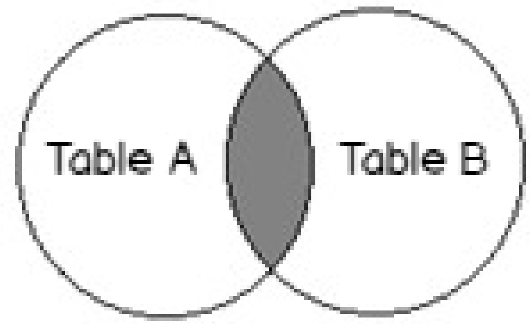
a Theta join where operator is = in all expressions.

- Examples:
  - All enrollments with their name, major, date and place of birth:
  - Name and major of all enrollments in Networks Intro

Equi Inner Join

$R \bowtie_F S$ where F is like $R.a_i = S.b_i$

$$R \bowtie_{A=B} S := \{r \cup s | r \in R \wedge s \in S \wedge r_{[A]} = s_{[B]}\}$$



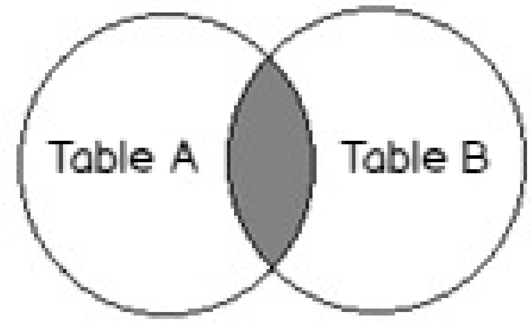
a Theta join where operator is = in all expressions

English	All enrollments with their name, major, date and place of birth
Relational Alg.	Enrollments $\bowtie_{\text{Enrollments.StudID=Students.StudID}}$ Students
SQL	SELECT * FROM Enrollments JOIN Students ON Enrollments.StudID = Students.StudID

Equi Inner Join

R $\bowtie_F$ S where F is like $R.a_i = S.b_i$

$$R \bowtie_{A=B} S := \{r \cup s | r \in R \wedge s \in S \wedge r_{[A]} = s_{[B]}\}$$



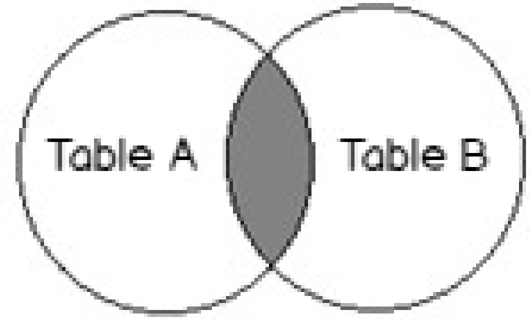
a Theta join where operator is = in all expressions

English	Name and major of all enrollments in Networks course
Relational Alg.	$\Pi_{\text{StudName, Major}}(\sigma_{\text{CourseTitle='Networks'}}($ Enrollments $\bowtie_{\text{Enrollments.StudID=Students.StudID}}$ Students))
SQL	SELECT StudName, Major FROM Enrollments INNER JOIN Students ON Enrollments.StudID = Students.StudID WHERE CourseTitle = 'Networks'

Natural join

R $\bowtie$ S

$$R \bowtie S := \{r \cup s_{[C_1,\ldots,C_n]} | r \in R \wedge s \in S \wedge r_{[B_1,\ldots,B_n]} = s_{[B_1,\ldots,B_n]}\}$$



The natural join is an equi-join of two relations R and S over all common attributes. One occurrence of each common attribute is removed from the result.

INNER JOIN – common columns are duplicated

NATURAL JOIN – common columns are included only once

English	Name and major of all enrollments in Networks course
Relational Alg.	$\Pi_{\text{StudName, Major}}(\sigma_{\text{CourseTitle='Networks'}}$ (Enrollments $\bowtie$ Students))
SQL	SELECT StudName, Major FROM Enrollments NATURAL JOIN Students WHERE CourseTitle = 'Networks'

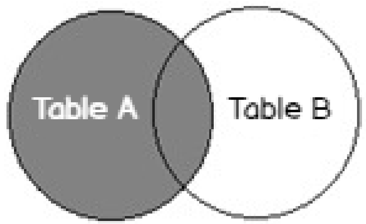
(*) SQL NATURAL JOIN is not supported by all DBMSes

Extensions to Relational Algebra

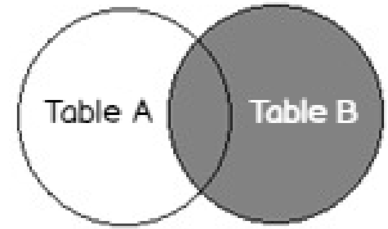
- Left / Right Outer join
- Full Outer join
- Left / Right Semi join
- Left / Right Anti join

Left / Right Outer Join

$R ⟕ S$



$R ⟖ S$



The left outer join is a join in which tuples from R that do not have matching values in the common columns of S are also included in the result relation

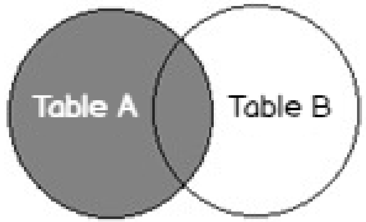
Missing values in the second relation (S) are set to null.

Preserves tuples that would have been lost with other types of join.

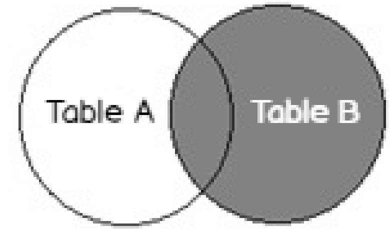
Similarly, the right outer join preserves tuples from the right-hand side relation.

Left / Right Outer Join

R ⟕ S



R ⟖ S

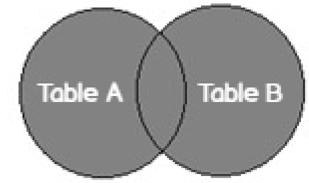


English	All students and for each one what courses he/she enrolled into, or NULL if none
Relational Alg.	Students ⟕ Enrollments
SQL	SELECT * FROM Students LEFT JOIN Enrollments ON Students.StudID = Enrollments.StudID

English	All courses offered by CS department and for each course the list of enrolled students, or NULL if none
Relational Alg.	$\sigma_{\text{Department = 'CS'}}$ (Enrollments ⟖ Courses)
SQL	SELECT * FROM Enrollments RIGHT JOIN Courses ON Enrollments.CourseTitle = Courses.CourseTitle WHERE Department = 'CS'

Full Outer Join

R ⟗ S

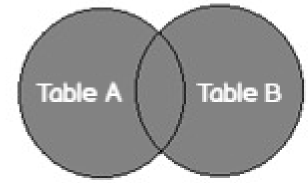


The result of the full outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in S that have no matching tuples in R and tuples in R that have no matching tuples in S in their common attribute names.

Missing values are set to null.

Full Outer Join

R ⟗ S



The result of the full outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuples in S that have no matching tuples in R and tuples in R that have no matching tuples in S in their common attribute names.

Missing values are set to null.

English	All students and all courses offered by the department of their major
Relational Alg.	Students ⟗ Courses
SQL	SELECT * FROM Students FULL JOIN Courses ON Students.Major = Courses.Department

Left / Right Semi Join

R $\ltimes$ S $\qquad R \ltimes S := \{r \mid r \in R \wedge s \in S \wedge r_{[B_1,\ldots,B_n]} = s_{[B_1,\ldots,B_n]}\}$

Returns a relation that contains only the columns of R; only the tuples of R that participate in the join with S will be returned and only **once**, regardless how many times matched in S.

$$R \ltimes S = \pi_{A1,\ldots,An}(R \bowtie S)\text{, where } R(A_1, \ldots, A_n)$$

Table 1

	ProductTypeId	ProductName
1	1	Hard disk
2	1	Hard disk
3	2	SSD disk
4	3	Flash Memory
5	4	EPROM
6	5	NULL
7	6	Floppy Disk
8	7	RAM Memory

Table 1
LEFT SEMI JOIN Table 2
ON(ProductName = Name)

Table 2

	ProductId	Name	StorageSizeGB
1	1	Hard disk	250
2	1	SSD disk	120
3	2	SSD disk	240
4	3	Flash Memory	8
5	4	EPROM	0
6	5	NULL	NULL

Table 1 subset (result-set)

	ProductTypeId	ProductName
1	1	Hard disk
2	1	Hard disk
3	2	SSD disk
4	3	Flash Memory
5	4	EPROM

Source: https://sqlchitchat.com/sqldev/tsql/semi-joins-in-sql-server/

Left / Right Semi Join

R ⋉ S $\quad R \ltimes S := \{r | r \in R \wedge s \in S \wedge r_{[B_1,\ldots,B_n]} = s_{[B_1,\ldots,B_n]}\}$

Returns a relation that contains only the columns of R; only the tuples of R that participate in the join with S will be returned and only **once**, regardless how many times matched in S.

$R \ltimes S = \pi_{A1,\ldots,An}(R \bowtie S)$, where $R(A_1, \ldots, A_n)$

Variant 1:

Self-contained, multi-valued subquery

```
SELECT *
FROM R
WHERE R.A IN (SELECT S.B FROM S)
```

Variant 2:

Correlated, multi-valued subquery

```
SELECT *
FROM R
WHERE R.A IN (SELECT S.B FROM S WHERE R.A = S.B)
```

Variant 3:

```
SELECT R.* FROM R NATURAL JOIN S /*when R and S share common cols*/
```

Left / Right Semi Join

R ⋉ S $\quad R \ltimes S := \{r | r \in R \land s \in S \land r_{[B_1,\ldots,B_n]} = s_{[B_1,\ldots,B_n]}\}$

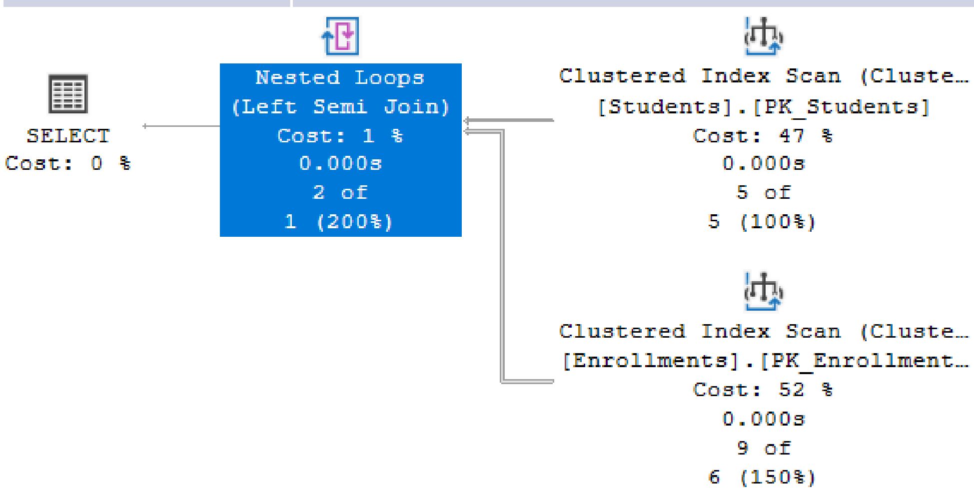
Returns a relation that contains only the columns of R; only the tuples of R that participate in the join with S will be returned and only **once**, regardless how many times matched in S.

English	Full details of students who are accepted in the Networks course
Relational Alg.	Students ⋉ ($\sigma_{\text{CourseTitle='Networks' AND Decision=T}}$ (Enrollments))
SQL	SELECT * FROM Students WHERE StudID IN (SELECT StudID FROM Enrollments WHERE CourseTitle='Networks' AND Decision=TRUE)

Left / Right Semi Join

R $\ltimes$ S $$R \ltimes S := \{r | r \in R \wedge s \in S \wedge r_{[B_1,\ldots,B_n]} = s_{[B_1,\ldots,B_n]}\}$$

English	Full details of students who are accepted in the Networks course
Relational Alg.	Students $\ltimes$ ($\sigma_{\text{CourseTitle='Networks' AND Decision=TRUE}}$ (Enrollments))
SQL	SELECT * FROM Students WHERE StudID IN (SELECT StudID FROM Enrollments WHERE Enrollments.StudID = Students.StudID AND CourseTitle='Networks' AND Decision=TRUE)



Left / Right Anti Join

$R \triangleright S$ $\quad\quad R \triangleright S = R - (R \ltimes S)$

Does the opposite of semi join, i.e. returns the rows in R that do not have at least one matching row in table S.

Variant 1:

```
SELECT *
FROM R
WHERE R.A NOT IN (SELECT S.B FROM S)
```

Variant 2:

```
SELECT *
FROM R
WHERE R.A NOT IN (SELECT S.B FROM S WHERE R.A = S.B)
```

Variant 3:

SELECT *
FROM R
WHERE NOT **EXISTS** (SELECT S.B FROM S WHERE R.A = S.B)

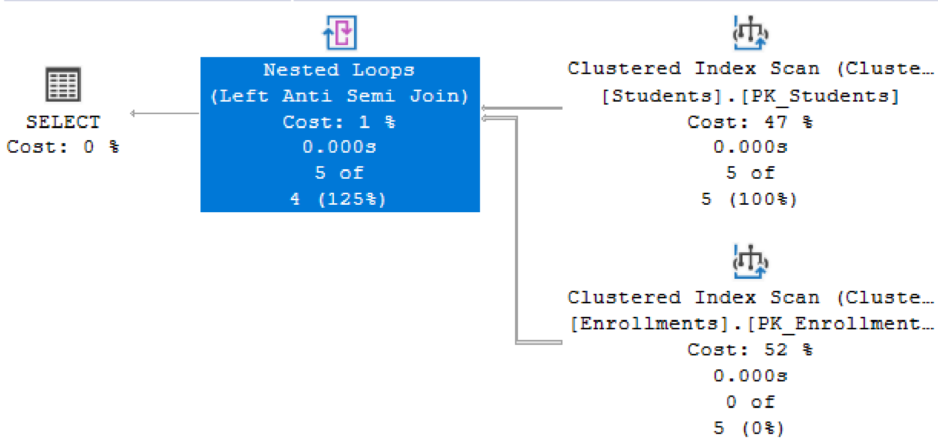
Variant 4:

```
SELECT R.A FROM R
EXCEPT
SELECT S.B FROM S
```

Left / Right Anti Join

$R \triangleright S$ $\qquad R \triangleright S = R - (R \ltimes S)$

English	Full details of students who are not accepted in the Networks course
Relational Alg.	Students $\triangleright (\sigma_{\text{CourseTitle='Networks' AND Decision=TRUE}}$ (Enrollments))
SQL	SELECT * FROM Students WHERE NOT EXISTS (SELECT StudID FROM Enrollments WHERE Enrollments.StudID = Students.StudID AND CourseTitle='Networks' AND Decision=TRUE)

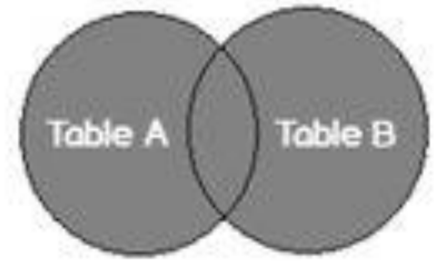


Set Operations

- UNION
- DIFFERENCE
- INTERSECTION

Set Union

- R u S $R \cup S := \{t | t \in R \vee t \in S\}$

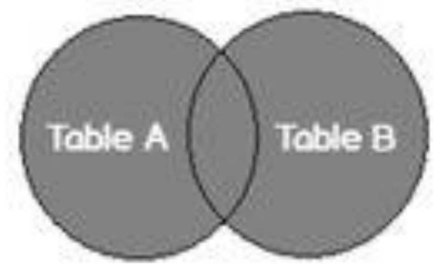


The union of two relations R and S with I and J tuples respectively, is obtained by concatenating them into one relation with a maximum of I + J tuples, duplicates being eliminated.

- R and S must be union compatible
  - they have the same number of attributes / columns
  - Corresponding attributes have the matching domains (same data type with the same length)

Set Union

- R u S $R \cup S := \{t | t \in R \vee t \in S\}$



The union of two relations R and S with I and J tuples respectively, is obtained by concatenating them into one relation with a maximum of I + J tuples, duplicates being eliminated.

- SQL Implementation

UNION – removes duplicated rows

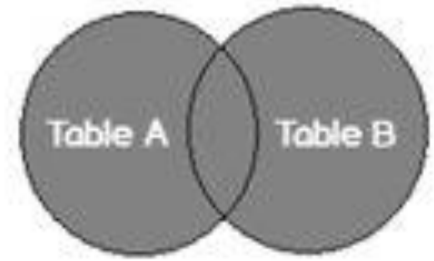
```
SELECT * FROM R
UNION
SELECT * FROM S
```

UNION ALL – preserves duplicated rows

```
SELECT * FROM R
UNION ALL
SELECT * FROM S
```

Set Union

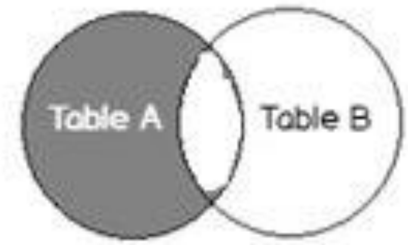
- R u S $R \cup S := \{t | t \in R \vee t \in S\}$



The union of two relations R and S with I and J tuples respectively, is obtained by concatenating them into one relation with a maximum of I + J tuples, duplicates being eliminated.

English	List of all departments and majors
Relational Alg.	$\Pi_{\text{Department}}$(Courses) $\cup$ Π_{Major}(Students)
SQL	SELECT Department FROM Courses UNION SELECT Major FROM Students

Set Difference



- R – S

Defines a relation consisting of the tuples that are in relation R, but not in S.

- R and S must be union compatible.
- SQL Implementation
  - MS SQL Sever, PostgreSQL – EXCEPT
  - Oracle – MINUS
  - MySQL & others – not supported

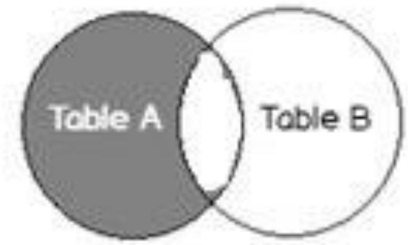
Recommended

```
SELECT R.*
FROM R
  LEFT JOIN S ON R.ID=S.ID
WHERE S.ID IS NULL
```

```
SELECT *
FROM R
WHERE NOT EXISTS
(SELECT * FROM S
WHERE R.ID = S.ID)
```

```
SELECT *
FROM R
WHERE NOT IN (SELECT
* FROM S WHERE R.ID =
S.ID)
```

Set Difference



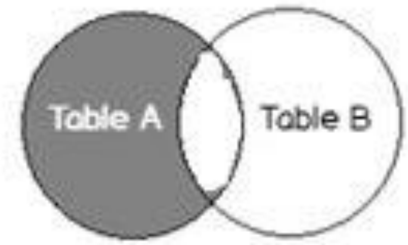
- R – S

Defines a relation consisting of the tuples that are in relation R, but not in S.

- R and S must be union compatible.

English	IDs of students who didn't apply for any course
Relational Alg.	Π_{StudID} (Students) - Π_{StudID}(Enrollments)
SQL	SELECT StudID FROM Students EXCEPT SELECT StudID FROM Enrollments

Set Difference



- R – S

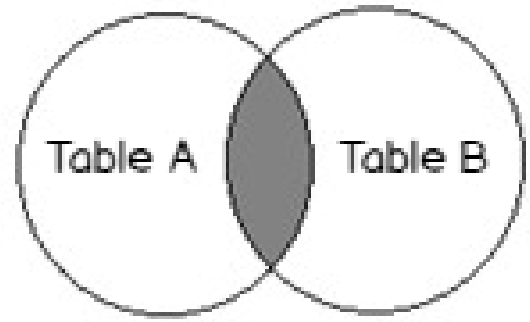
Defines a relation consisting of the tuples that are in relation R, but not in S.

- R and S must be union compatible.

English	IDs of students who didn't apply for any course
Relational Alg.	Π_{StudID} (Students) - Π_{StudID}(Enrollments)
SQL	SELECT Students.StudID FROM Students LEFT JOIN Enrollments ON Enrollments.StudID = Students.StudID WHERE Enrollments.StudID IS NULL

Set Intersection

- $R \cap S$



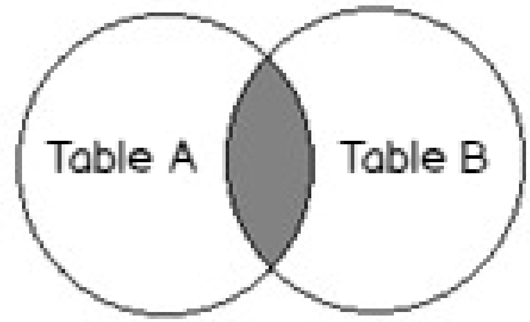
- Consists of the set of all tuples that are in both R and S.

- R and S must be union-compatible.

- Not additional expressiveness to Relational Algebra
  - $R \cap S = R - (R - S)$
  - $R \cap S = R \bowtie S$

Set Intersection

- $R \cap S$



- Consists of the set of all tuples that are in both R and S.

English	All nouns that are both department names and majors
Relational Alg.	$\Pi_{\text{Department}}(\text{Courses}) \cap \Pi_{\text{Major}}(\text{Students})$
SQL	SELECT Department FROM Courses INTERSECT SELECT Major FROM Students
SQL	SELECT DISTINCT Department FROM Courses INNER JOIN Students ON Courses.Department = Students.Major

Exercise

- Write the relational algebra expression and its SQL implementation that returns the IDs and names of students who did not apply to any course.

Exercise

- Write the relational algebra expression and its SQL implementation that returns the IDs and names of students who did not apply to any course.

English	IDs and names of students who didn't apply for any course
Relational Alg.	$\Pi_{\text{StudID, StudName}}((\Pi_{\text{StudID}}(\text{Students}) - \Pi_{\text{StudID}}(\text{Enrollments})) \bowtie \text{Students})$
SQL	SELECT Students.StudID, StudNames FROM Students LEFT JOIN Enrollments ON Enrollments.StudID = Students.StudID WHERE Enrollments.StudID IS NULL

Exercise

Which of the following English sentences describes the result of the following expression?

$\pi_{CourseTitle}(\text{Courses}) - \pi_{CourseTitle}($

$\text{Enrollments} \bowtie (\pi_{StudID}(\sigma_{PoB='Timisoara'}(\text{Students})) \cap \pi_{StudID}(\sigma_{Decision=TRUE}(\text{Enrollments}))))$

a) All courses where all Students either were born in Timisoara or were accepted in any course

b) All courses with no Timisoara-born students who were accepted at any course

c) All courses with no Timisoara-born students or rejected students

Exercise

Which of the following English sentences describes the result of the following expression?

$\pi_{CourseTitle}(Courses)$ - $\pi_{CourseTitle}($

$Enrollments \bowtie (\pi_{StudID}(\sigma_{PoB=\text{'Timisoara'}}(Students)) \cap \pi_{StudID}(\sigma_{Decision=TRUE}(Enrollments)))))$

a) All courses where all Students either were born in Timisoara or were accepted in any course

b) All courses with no Timisoara-born students who were accepted at any course

c) All courses with no Timisoara-born students or rejected students

Other Extensions to Relational Algebra

- Division

- Extended projection
  - Aggregations
  - Groupings

Division

R $\div$ S $\quad R \div S := \pi_{R'}(R) - \pi_{R'}((\pi_{R'}(R) \times S) - R)$

Defines a relation over the attributes C that consists of the set of tuples from R that match the combination of every tuple in S.

$$T_1 \leftarrow \Pi_C(R)$$

$$T_2 \leftarrow \Pi_C((T_1 \times S) - R)$$

$$T \leftarrow T_1 - T_2$$

Example: Identify all students who enrolled to all courses offered by CS department.

Division

R $\div$ S $\quad R \div S := \pi_{R'}(R) - \pi_{R'}((\pi_{R'}(R) \times S) - R)$

Defines a relation over the attributes C that consists of the set of tuples from R that match the combination of every tuple in S.

$$T_1 \leftarrow \Pi_C(R)$$

$$T_2 \leftarrow \Pi_C((T_1 \times S) - R)$$

$$T \leftarrow T_1 - T_2$$

Example: Identify all students who enrolled to all courses offered by CS department.

$\Pi_{\text{StudID, CourseTitle}}$ (Enrollments) $\div$ $\Pi_{\text{CourseTitle}}$ ($\sigma_{\text{Dept='CS'}}$ (Courses))

There is No equivalent SQL command! Have a look at below for details

http://gregorulm.com/relational-division-in-sql-the-easy-way/

Aggregate

$\Im_{AL}(R)$	Applies the aggregate function list, AL, to the relation R to define a relation over the aggregate list. AL contains one or more (<aggregate_function>, <attribute>) pairs.

The main aggregate functions are:
- COUNT – returns the number of values in the associated attribute.
- SUM – returns the sum of the values in the associated attribute.
- AVG – returns the average of the values in the associated attribute.
- MIN – returns the smallest value in the associated attribute.
- MAX – returns the largest value in the associated attribute.

English	Find the number of students born in Timisoara
Relational Alg.	$\Im_{\text{COUNT StudId}}(\sigma_{\text{PoB='Timisoara'}}(\text{Students}))$
SQL	SELECT COUNT(StudID) AS StudCount FROM Student WHERE PoB='Timisoara'

Grouping

$_{GA}\Im_{AL}(R)$	Groups the tuples of relation R by the grouping attributes, GA, and then applies the aggregate function list AL to define a new relation. AL contains one or more (<aggregate_function>, <attribute>) pairs. The resulting relation contains the grouping attributes, GA, along with the results of each of the aggregate functions.

$$_{a_1, a_2, \ldots, a_n}\Im_{<A_p\, a_p>, <A_q\, a_q>, \ldots, <A_z\, a_z>}(R)$$

$$\gamma_{F(X);A}(R) := \bigcup_{t \in R} \gamma_{F(X);\emptyset}(\sigma_{A=t.A}(R))$$

English	Find the number of students born in each city
Relational Alg.	$\rho_{R(PlaceOfBirth,\ NbStudents)}\ (_{PoB}\Im_{COUNT\ StudId}(Students))$
SQL	SELECT PoB AS ‘Place of Birth’, COUNT(*) AS ‘Nb of students’ FROM Students GROUP BY PoB

The Anatomy of the SELECT statement

```
SELECT [TOP n] [DISTINCT]
    column [AS alias]
    , AGG_FUNC(column_or_expression) [AS alias]
    , ...
FROM mytable
JOIN another_table ON mytable.column = another_table.column
WHERE
    constraint_expression
GROUP BY
    column
HAVING
    constraint_expression
ORDER BY
    column ASC/DESC
LIMIT
    count
OFFSET
    skip
```

Projection

Join (equi)

Selection

Logical Query Processing Order

The logical query processing order is the logical order in which the clauses that make up a SELECT statement are processed.

```
SELECT [TOP n] [DISTINCT]
    column [AS alias]
    , AGG_FUNC(column_or_expression) [AS alias]
    , …
FROM mytable
JOIN another_table ON mytable.column = another_table.column
WHERE
    constraint_expression
GROUP BY
    column
HAVING
    constraint_expression
ORDER BY
    column ASC/DESC
LIMIT
    count
OFFSET
    skip
```

FAST	FROM & JOINs
WALKING	WHERE
GIANTS	GROUP BY
HAVE	HAVING
SMELLY	SELECT [DISTINCT]
ODOR	ORDER BY

Logical Query Processing Order

To add to the above list there are two keywords used in the SELECT clause that are processed after the ORDER BY when they are present.
They are logically processed in the following order:

DISTINCT - Removes all duplicate records after the data has been ordered.

TOP - Returns the TOP number or percentage of rows after the data has been ordered and duplicates have been removed when DISTINCT is present.

OFFSET/LIMIT – Are applied after DISTINCT and ORDER BY

Subqueries

```
SELECT [TOP n] [DISTINCT]  -- < OUTER QUERY
    column [AS alias]
    , AGG_FUNC(column_or_expression) [AS alias]
    , …
FROM mytable
JOIN another_table ON mytable.column = another_table.column
WHERE
    Attr = (SELECT …)              -- < INNER QUERY
```

Subquery can be

- SELF-CONTAINED – has no references to the outer query
- CORRELATED – has references to the outer query and is dependent on it.

Subquery can be

- SINGLE-VALUES (SCALAR) – returns a scalar value
- MULTI-VALUED – returns a set

Mapping relational operators to SQL

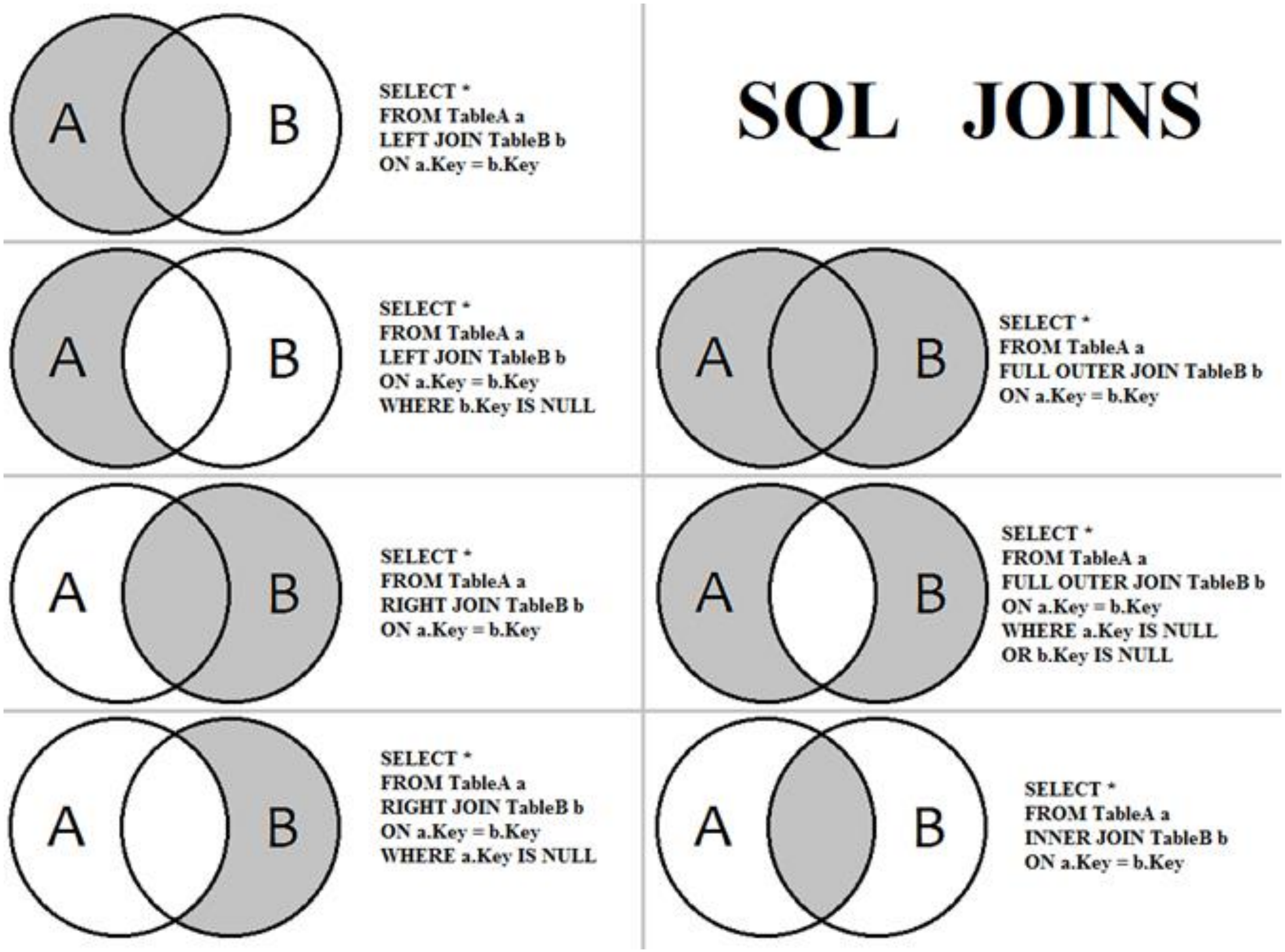
Relational operator	SQL support
$\sigma_{predicate}(R)$	SELECT * FROM R WHERE predicate
$\Pi_{col1,\ldots,coln}(R)$	SELECT col1, …, coln FROM R
$\rho_{R(A_1, \ldots, A_n)}(Exp)$	AS (e.g. col1 AS A1or Table1 AS T1)
R ∪ S	R UNION S R UNION ALL S
R – S	R EXCEPT S (or R MINUS S) SELECT R.* FROM R LEFT JOIN S ON R.ID=S.ID WHERE S.ID IS NULL
R ∩ S	R INTERSECT S SELECT DISTINCT * FROM R (INNER | NATURAL) JOIN S
R x S	SELECT * FROM R, S SELECT * FROM R CROSS JOIN S

Mapping relational operators to SQL

Relational operator	SQL support
$R \bowtie_F S$	SELECT * FROM R, S WHERE F
$R \bowtie S$	SELECT * FROM R NATURAL JOIN S SELECT * FROM R INNER JOIN S ON R.CA = S.CA
$R ⟕ S$	SELECT * FROM R LEFT OUTER JOIN S ON R.CA = S.CA
$R ⟗ S$	SELECT * FROM R FULL OUTER JOIN S ON R.CA = S.CA
$R \ltimes S$	SELECT * FROM R WHERE R.A IN (SELECT S.B FROM S)
$R \triangleright S$	SELECT * FROM R WHERE NOT EXISTS (SELECT S.B FROM S WHERE R.A = S.B)
$R \div S$	
$\mathbf{S}_{AL}(R)$	SELECT <AL> FROM R
${}_{GA}\mathbf{G}_{AL}(R)$	SELECT GA1, …, GAn <AL> FROM R GROUP BY GA1, …, GAn

Remark: CA in above SELECT statements stands for Common Attributes and is the list of one or more common attributes shared by R and S tables.

Joins Explained



Time for a Quiz

Some Exercises

```sql
-- Exercise: List the id, name, surname, date of
birth, place of birth, major and any unconfirmed
enrollment they may have
SELECT S.StudID AS ID,
LEFT(StudName, CHARINDEX(' ', StudName)) AS NUME,
RIGHT(StudName, LEN(StudName) - CHARINDEX(' ',
StudName)) AS PRENUME,
FORMAT(DoB, 'dd MMMM yyyy') AS "DATA NASTERII",
PoB AS "LOCUL NASTERII",
Major AS SPECIALIZARE,
EC.*
FROM Students S
LEFT JOIN (SELECT E.*, C.Department, C.Credits
   FROM Enrollments E
   INNER JOIN Courses C ON E.CourseTitle =
C.CourseTitle) EC ON S.StudID = EC.StudID
WHERE Accepted IS NULL
ORDER BY NUME, PRENUME
```

```sql
-- Exercise: List the name of the students and
their total unconfirmed credits sorted desc by
total unconfirmed credits, for all students with
at least 2 unconfirmed enrollments
SELECT StudName, SUM(EC.Credits) AS
UnconfirmedCredits
FROM Students S
LEFT JOIN (SELECT E.*, C.Department, C.Credits
   FROM Enrollments E
   INNER JOIN Courses C ON E.CourseTitle =
C.CourseTitle) EC ON S.StudID = EC.StudID
WHERE Accepted IS NULL
GROUP BY StudName
HAVING COUNT(*) > 1
ORDER BY UnconfirmedCredits DESC

-- Exercise: Find duplicate student names
SELECT
  LEFT(StudName, CHARINDEX(' ', StudName)) AS
NUME,
  COUNT(*) AS DUPLICATE
FROM Students S
GROUP BY LEFT(StudName, CHARINDEX(' ', StudName))
HAVING COUNT(*) > 1
ORDER BY NUME
```

When a DBMS is relational?

- The 12 + 1 rules of Codd

- Foundational rules
- Structural rules
- Integrity rules
- Data manipulation rules
- Data independence rules

Foundational rules

- Rule 0: The system must be able to manage databases entirely through its relational capabilities

- Rule 12 (non-subversion): If a relational system has a low level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time)

Structural rules

- Rule 1 (information representation): All information is represented explicitly at the logical level and in exactly one way – by values in tables
- Rule 6 (view updating): All views that are theoretically updatable are also updateable by the system

Integrity rules

- Rule 3 (systematic treatment of null values): Null values are supported for representing missing information and inapplicable information in a systematic way, independent of data type

- Rule 10 (integrity independence): Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in applications

Data manipulation rules

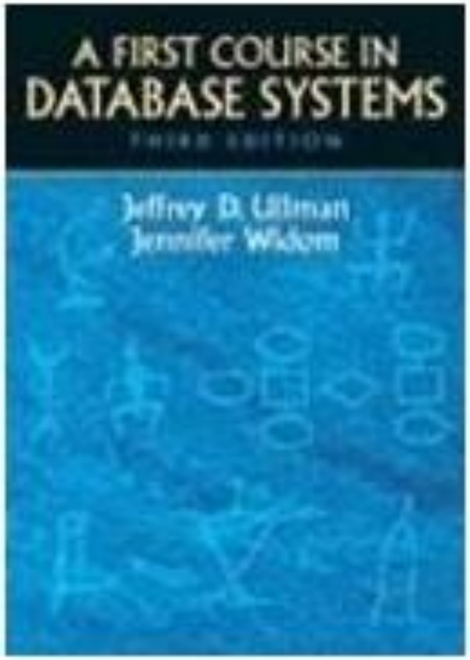
- Rule 2 (guaranteed access): Each and every atomic value in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name
- Rule 4 (dynamic online catalog based on the relational model): The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to regular data
- Rule 5 (comprehensive data sublanguage): There must be at least one language whose statements can express all of the following items: data definition, view definition, data manipulation, integrity constraints, authorization, transaction boundaries
- Rule 7 (high level insert, update, delete): The capability of handling a base relation or a derived relation as a single operand applies not only to data retrieval but also to the insertion, update, and deletion of data

Data independence rules

- Rule 8 (physical data independence): Apps remain logically unimpaired whenever any changes are made in either storage representations or access methods
- Rule 9 (logical data independence): Apps remain logically unimpaired when information-preserving changes of any kind that permit unimpairment are made to base tables
- Rule 11 (distribution independence): The DML must enable apps to remain logically the same whether and whenever data are physically centralized or distributed

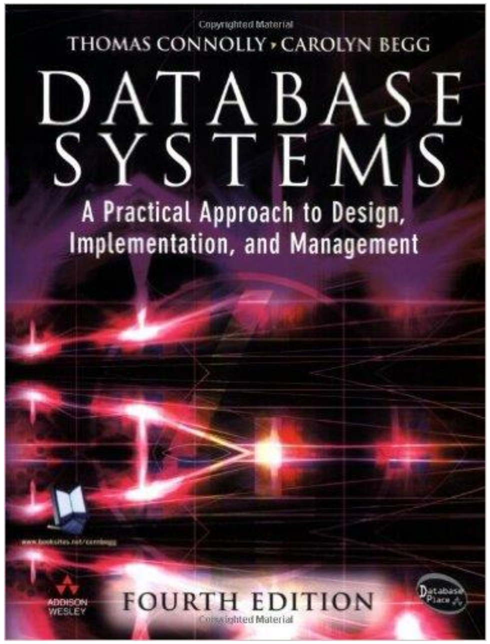
References

Teoria generala a bazelor de date, I. Despi, G. Petrov, R. Reisz, A. Stepan, Mirton, 2000
<u>Cap 5</u>



A First Course in Database Systems (3rd edition) by Jeffrey Ullman and Jennifer Widom, Prentice Hall, 2007

<u>Chapter 2</u>



Database Systems - A Practical Approach to Design, Implementation, and Management (4th edition) by Thomas Connolly and Carolyn Begg, Addison-Wesley, 2004
<u>Chapter 3 & 4</u>

References

- SQL multiple joins for beginners with examples.
  - https://www.sqlshack.com/sql-multiple-joins-for-beginners-with-examples
- Semi join, anti join and division in SQL Server
  - https://sqlchitchat.com/sqldev/tsql/semi-joins-in-sql-server/
- SQL Coding Style & Best Practices
  - https://www.red-gate.com/simple-talk/sql/t-sql-programming/sql-code-smells/
  - https://www.red-gate.com/simple-talk/sql/t-sql-programming/basics-good-t-sql-coding-style/
- Logical Joins vs Physical Joins
  - https://www.c-sharpcorner.com/blogs/physical-join-vs-logical-join-in-sql-server