Databases 2: Introduction to Database Management Systems

Ioan Dragan

Sept. 2020

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

History

1. 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
2. System R developed by IBM at San Jose Research Laboratory, California, late 1970s
   - Led to the development of SQL
   - Initiated the production of commercial RDBMs
3. INGRES (Interactive Graphics REtrieval System) at the University of California at Berkley.

Genealogy of Relational Database Management System

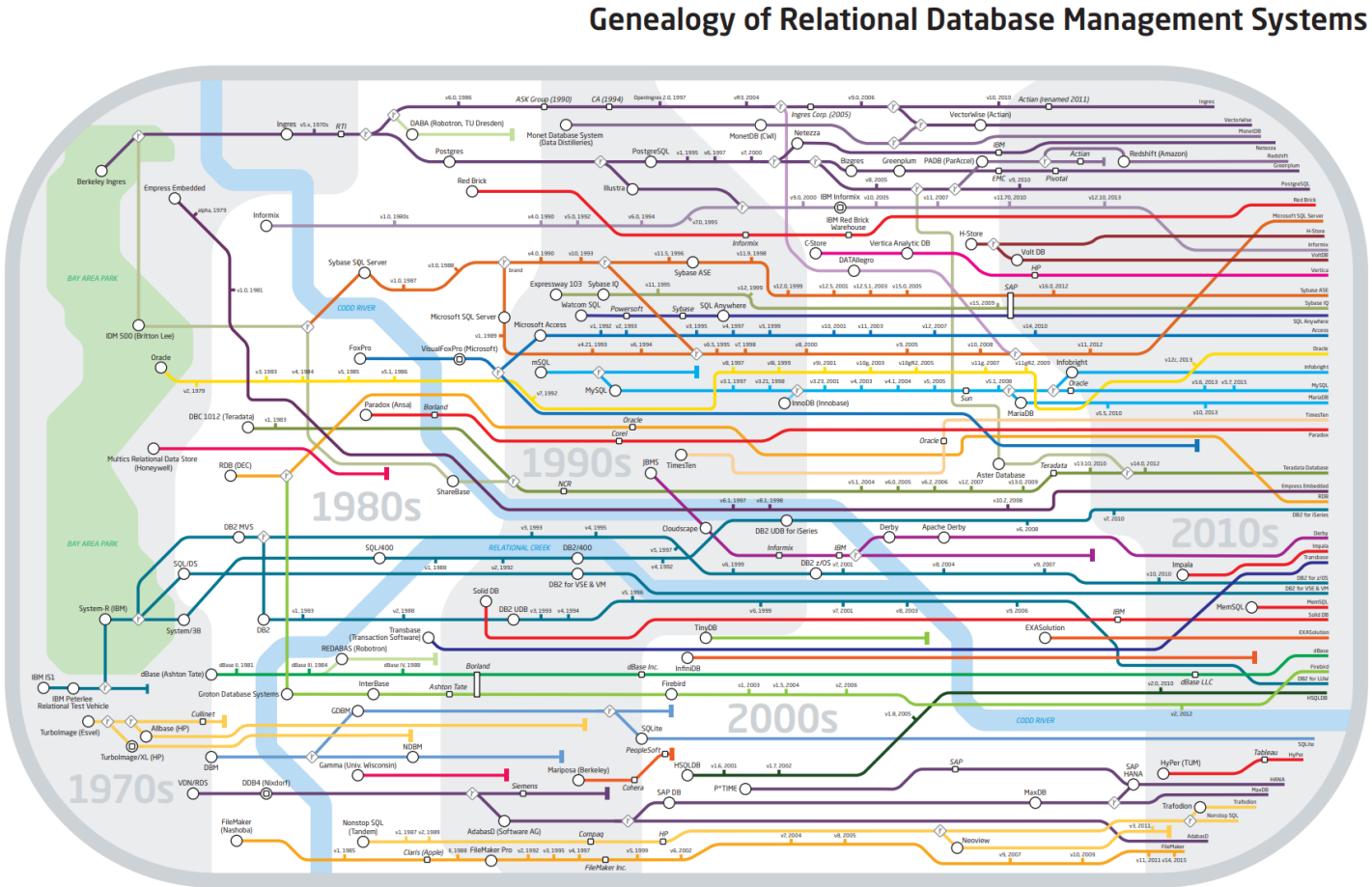


Figure: Source: `hhttps://hpi.de/naumann/projects/rdbms-genealogy.html`

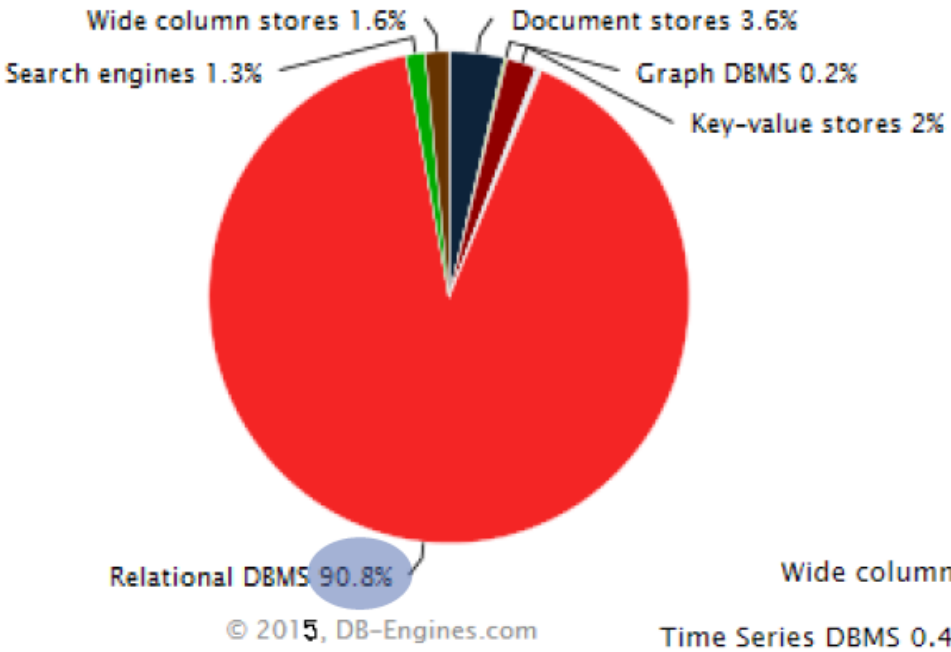
Popular models used by DBMS

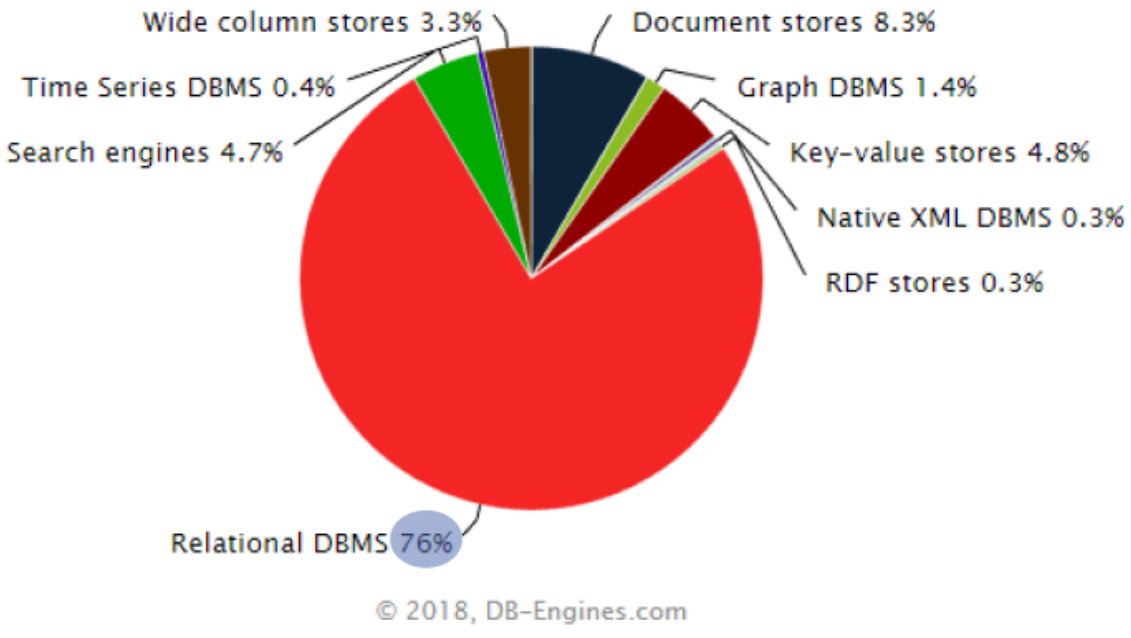
359 systems in ranking, October 2020

Rank Oct 2020	Rank Sep 2020	Rank Oct 2019	DBMS	Database Model	Score Oct 2020	Score Sep 2020	Score Oct 2019
1.	1.	1.	Oracle	Relational, Multi-model	1368.77	-0.59	+12.89
2.	2.	2.	MySQL	Relational, Multi-model	1256.38	-7.87	-26.69
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1043.12	-19.64	-51.60
4.	4.	4.	PostgreSQL	Relational, Multi-model	542.40	+0.12	+58.49
5.	5.	5.	MongoDB	Document, Multi-model	448.02	+1.54	+35.93
6.	6.	6.	IBM Db2	Relational, Multi-model	161.90	+0.66	-8.87
7.	↑ 8.	7.	Elasticsearch	Search engine, Multi-model	153.84	+3.35	+3.67
8.	↓ 7.	8.	Redis	Key-value, Multi-model	153.28	+1.43	+10.37
9.	9.	↑ 11.	SQLite	Relational	125.43	-1.25	+2.80
10.	10.	10.	Cassandra	Wide column	119.10	-0.08	-4.12

Figure: Source: `https://db-engines.com/en/ranking`

Most popular models by 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:
  1. Collection of relations
  2. Set of operators to act on the relation
  3. 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 at a given point in time of the database

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

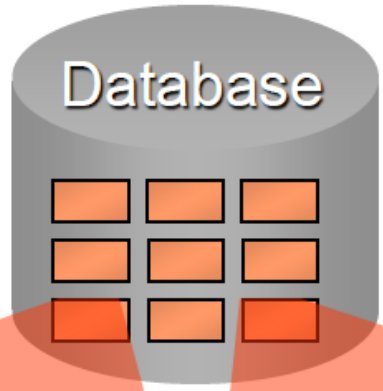


Table Name: EMP

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

FMI

Database relations

Relational schema = a relation name followed bu a set of attribute and domain name pairs

$$R = \{A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n\} \quad (1)$$

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 significance

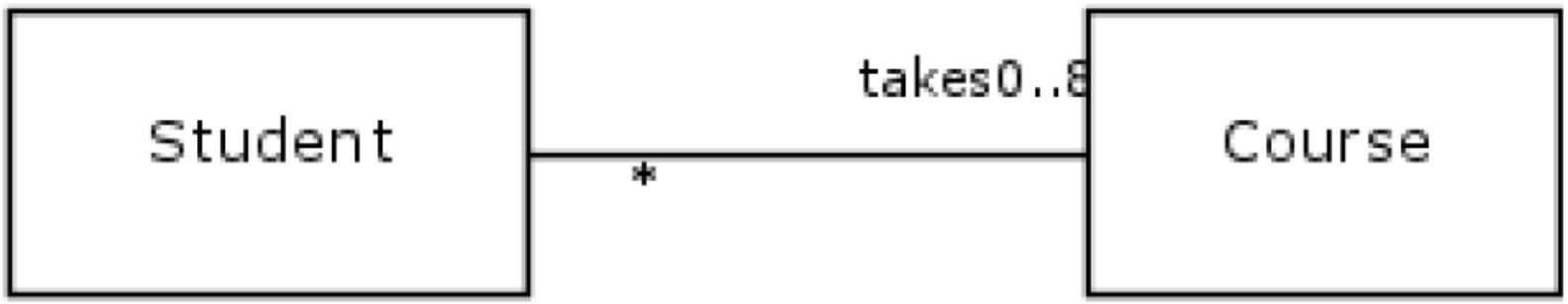
1NF - a relation satisfying these constraints

Running Example

Students should enroll in courses they want to attend. One student may enrolls in up to 8 courses. In order for one course to run it requires at least 10 enrolled students. As places in courses are limited, for each enrollment request there will be a decision associated whether the student is accepted or not in the course.
Courses are offered by different departments of the university, each course is uniquely identified by their title and each course is credited a fixed number of credits. Students may enroll to courses offered by different departments.

Example

Conceptual model



FMI

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
- 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
  - Irreducibility - no proper subset of K has the uniqueness property
- Primary key = a candidate key selected by the database designer to uniquely identify tuples within a relation
- Alternate key = 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 some (possibly the same) relation

Exercise

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

```
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)
```

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)
```

FMI

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)

Figure: Example from Database Systems - A practical approach to Design, Implementation, and Management (4^{th} edition)

FMI

Relational integrity

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

FMI

Relational Integrity

- Entity integrity in a base relation, no attribute of a primary key can be null.
- 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
- Structural Query Language (SQL) : de facto/implemented
- The query language also used for DML and DDL
- Some queries to pose, some more difficult
- Some easy to execute, others more difficult (expensive to compute)

FMI

Examples

- List name and date of birth of all students with major in CS
- Relational algebra: Formal
  $\Pi_{StudName,DoB}(\sigma_{Major='CS'}(Students))$
- Structured Query Language (SQL) - de facto/implemented

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

FMI

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
- **Bold** operators originally defined by **E.F. Codd** in 1970

Table name

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

Selection

$$\sigma_{predicate}(R)\text{: } \sigma_P(R) := \{t | t \in R \wedge R(t) = true\}$$

- 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
- Example:
  - Students with Major in CS

Selection

$$\sigma_{predicate}(R)\text{: } \sigma_P(R) := \{t | t \in R \wedge R(t) = true\}$$

- 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
- Example:
  - Students with Major in CS: $\sigma_{Major='CS'}(Students)$

FMI

Selection

$$\sigma_{predicate}(R)\text{: } \sigma_P(R) := \{t | t \in R \wedge R(t) = true\}$$

- 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
- Example:
  - Students with Major in CS: $\sigma_{Major='CS'}(Students)$
  - Students accepted in Database course

Selection

$$\sigma_{predicate}(R)\text{: } \sigma_P(R) := \{t | t \in R \wedge R(t) = true\}$$

- 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
- Example:
  - Students with Major in CS: $\sigma_{Major='CS'}(Students)$
  - Students accepted in Database course: $\sigma_{CourseTitle='Databases' \wedge Decision=TRUE}(Enrollments)$

FMI

Projection

$$\Pi_{col1,col2,...,coln}(R) : \pi_{\beta}(R) := \{t_{\beta} | t \in R\}$$

- 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.
- Example:
  - Name and major of all students:

FMI

Projection

$$\Pi_{col1,col2,...,coln}(R) : \pi_{\beta}(R) := \{t_{\beta} | t \in R\}$$

- 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.
- Example:
  - Name and major of all students: $\Pi_{StudName,Major}(Students)$

Projection

$$\Pi_{col1,col2,...,coln}(R) : \pi_{\beta}(R) := \{t_{\beta} | t \in R\}$$

- 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.
- Example:
  - Name and major of all students: $\Pi_{StudName,Major}(Students)$
- Remark:
  - In Relation Algebra, duplicates are ELIMINATED (set theory)
  - In SQL, duplicates are not!!! *rightarrow* in order to eliminate there is `SELECT DISTINCT` command;

FMI

Selection and projection

Examples:

- Name and date of birth of students with Major in CS
- Course title and number of credits of all courses offered by CS department

Selection and projection

Examples:

- Name and date of birth of students with Major in CS

  $\Pi_{StudName,DoB}(\sigma_{Major='CS'}(Students)$
- Course title and number of credits of all courses offered by CS department

  $\Pi_{CourseTitle,Credits}(\sigma_{Department='CS'}(Courses))$

FMI

Assignment statements

- Complex queries may be broken down into simpler expressions
- Example:
  $\Pi_{StudName,DoB}(\sigma_{Major='CS'}(Students))$
  is equivalent to
  $R_1 = \sigma_{Major='CS'}(Students)$
  $R_2 := \Pi_{StudName,DoB}(R_1)$

FMI

Cartesian/ Cross-Product

$$R \times S := \{(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m) | (a_1, a_2, \ldots, a_n) \in R \wedge (b_1, b_2, \wedge, b_m) \in S\}$$

- Returns a new relation that is the concatenation of every tuple of relation R with each tuple of realtion S
- Schemas are unified
- How many tuples in the Cartesian product $R \times S$?
- Example: Name and Major of students accepted in English course.

$$\Pi_{StudName, Major}(\sigma_{Students.StudId=Enrollment.StudID \wedge CourseTitle=English \wedge Decision=TRUE}(Students \times Enrollments))$$

Rename

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

- Usage: Disambiguation in self-joins
- Example: Pairs of courses offered by the same department

$\sigma_{D_1=D_2}(\rho_{C_1(CT1,D1,C1)}(Courses) \times \rho_{C_2(CT2,D2,C2)}(Courses))$

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?

1. $\Pi_{StudName, Major}(\sigma_{Students.StudId=Enrollment.StudID}(\sigma_{PoB=Timisoara}(Students) \times \sigma_{CourseTitle-Databases \wedge Decision=False}(Enrollments))$
2. $\Pi_{StudName, Major}(\sigma_{Students.StudID=StudID \wedge PoB=Timisoara \wedge CourseTitle=DB \wedge Decision=False}(Students \times \Pi_{StudentID, CourseTitle, Decision}(Enrollments)))$
3. $\sigma_{Students.StudId=Enrollment.StudID}(\Pi_{StudName, Major}(\sigma_{PoB=Timisoara}(Students \times \sigma_{CourseTitle=DB \wedge Decision=FALSE}(Enrollments))))$

Join Operations

- Typically we only need a susbset of the Cartesian product
- Types of join:
  - Theta join
  - Equi join
  - Natural join
- No additional power to **Relational Algebra** as there are shortened forms of other expressions.

Theta join

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

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:

$$<, >, \leq, \geq, =, <>$$

A Theta join is a shorthand form of:

$$R \bowtie_F S = \sigma_F(R \times S)$$

FMI

Equi Join

$R \bowtie_F S$ where F is like $R.a_i = S.b_i$
$R \bowtie_{A \Theta B} S := \{r \cup s | r \in R \wedge s \in S \wedge r_{[A]} = s_{[B]}\}$

A **Equi Join** is a Theta join where the operator is $=$ in all expressions.
Example:

- All enrollments with their name, major, date and place of birth

Equi Join

$R \bowtie_F S$ where F is like $R.a_i = S.b_i$
$R \bowtie_{A\Theta B} S := \{r \cup s | r \in R \wedge s \in S \wedge r_{[A]} = s_{[B]}\}$

A **Equi Join** is a Theta join where the operator is $=$ in all expressions.
Example:

- All enrollments with their name, major, date and place of birth
  $R_1 = Enrollment \bowtie_{Enrollment.StudentID=Students.StudID} Students$

FMI

Equi Join

$R \bowtie_F S$ where F is like $R.a_i = S.b_i$
$R \bowtie_{A\Theta B} S := \{r \cup s | r \in R \wedge s \in S \wedge r_{[A]} = s_{[B]}\}$

A **Equi Join** is a Theta join where the operator is $=$ in all expressions.
Example:

- All enrollments with their name, major, date and place of birth
  $R_1 = Enrollment \bowtie_{Enrollment.StudentID=Students.StudID} Students$
- Name and major of all enrollments in Networks intro.

FMI

Equi Join

$R \bowtie_F S$ where F is like $R.a_i = S.b_i$
$R \bowtie_{A \Theta B} S := \{r \cup s | r \in R \wedge s \in S \wedge r_{[A]} = s_{[B]}\}$

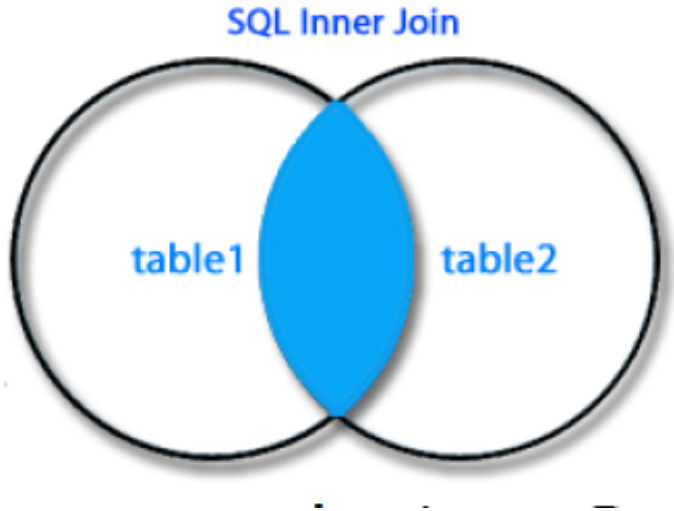
A **Equi Join** is a Theta join where the operator is $=$ in all expressions.
Example:

- All enrollments with their name, major, date and place of birth
  $R_1 = Enrollment \bowtie_{Enrollment.StudentID = Students.StudID} Students$
- Name and major of all enrollments in Networks intro.
  $R_2 = \Pi_{StudName, Major}(\sigma_{CourseTitle = "Networks"}(R_1))\}$

FMI

Natural Join

$R \bowtie S := \{r \cup s_{C1 \ldots Cn} | r \in R \wedge s \in S \wedge r_{[B1 \ldots Bn]} = s_{[B1 \ldots Bn]}\}$
The natural join is an equi-join of the two relations R and S over all common attributes. One occurrence of each common attribute is removed from the result.



FMI

Natural Join

$R \bowtie S := \{r \cup s_{C1 \ldots Cn} | r \in R \wedge s \in S \wedge r_{[B1 \ldots Bn]} = s_{[B1 \ldots Bn]}\}$
The natural join is an equi-join of the two relations R and S over all common attributes. One occurrence of each common attribute is removed from the result.

Example: Name and major of students enrolled in Networks.

$\Pi_{StudName,Major}(\sigma_{CourseTitle='Networks'}(Students \bowtie Enrollments))$

FMI

Set Union

- $R \cup S$: $R \cup S := \{t | t \in R \vee t \in S\}$
  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. The schema match, in other words, they have the same number of attributes with matching domains.
- Example: List of course titles and majors.

Set Union

- $R \cup S$: $R \cup S := \{t | t \in R \vee t \in S\}$
  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. The schema match, in other words, they have the same number of attributes with matching domains.
- Example: List of course titles and majors.

  $\Pi_{CourseTitle}(Courses) \cup \Pi_{Major}(Students)$

FMI

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.
- Example: IDs of students who did not enroll in any course

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.
- Example: IDs of students who did not enroll in any course
  $\Pi_{StudID}(Students) - \Pi_{StudID}(Enrollments)$

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.
- Example: IDs of students who did not enroll in any course
  $\Pi_{StudID}(Students) - \Pi_{StudID}(Enrollments)$
- IDs and names of students who did not enroll in any course.

Set Intersection

- $R \cap S$
- Consists of the set of all tuples that are both in R and in S.
- R and S must be union compatible
- Example: Nouns that are both Course titles and majors
- Not additional expressiveness to Relational Algebra:
  - $R \cap S = R - (R - S)$
  - $R \cap S = R \bowtie S$

Exercise

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

$$\Pi_{CourseTitle}(Courses) - \Pi_{CourseTitle}(Enrollment \bowtie (\Pi_{StudID}(\sigma_{PoB='Timisoara'}(Students)) \cap \Pi_{StudID}(\sigma_{Decision=TRUE}(Enrollments))))$$

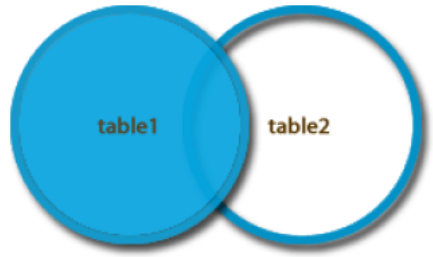
- All courses where all students either were born in Timisoara or were accepted in any course
- All courses with no Timisoara-born students who were accepted at any course
- All courses with no Timisoara-born students or rejected students

Extensions to Relational Algebra

- Left/Right outer join
- Full outer join
- Left/Right semi join
- Anti join
- Division
- Extended projection
  - Aggregations
  - Groupings

Left / Right Outer Join

$R \ltimes S$



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.

Example: List of all students and for each one what courses he/she enrolled into.

Students $\ltimes$ Enrollments

FMI

Full Outer Join

$R \overline{\bowtie} S$

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.

Left /Right Semi Join

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

Returns a relation that contains the tuples of R that participate in the join of R with S

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

Example: Full details of students who are accepted in the Networks course.

FMI

Left /Right Semi Join

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

Returns a relation that contains the tuples of R that participate in the join of R with S

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

Example: Full details of students who are accepted in the Networks course.

Student ⋉ ($\sigma_{\text{CourseTitle='Networks' AND Decision=T}}$ (Enrollments))

FMI

Anti Join

R ▷ S

Returns a relation that contains the tuples in R for which there is no tuple in S that is equal on their common attribute names

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

Example: Full details of students who are not accepted in the Networks course.

FMI

Division

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

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) ÷ $\Pi_{\text{CourseTitle}}$ ($\sigma_{\text{Dept='CS'}}$ (Courses))

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

FMI

[1] https://gregorulm.com/relational-division-in-sql-the-easy-way/

Aggregate

$\mathfrak{I}_{AL}(R)$ 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.

Example: Find the number of students born in Timisoara.

$$\mathfrak{I}_{\text{COUNT StudId}}(\sigma_{\text{PoB='Timisoara'}}(\text{Students}))$$

FMI

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.

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

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

Examples: Find the number of students born in each city

$\rho_{R(PlaceOfBirth,\ NbStudents)}\ (\ {}_{PoB}\Im_{COUNT\ StudId}\ (Students)$

FMI

SQL Implementation of relation model (short)

- 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.

FMI

SQL Implementation of relation model (short)

- Setting up Primary Key in different ways:

- While creating the table for single-attribute primary keys

```
CREATE TABLE Students (
    StudID int NOT NULL PRIMARY KEY,
...);
```

- While creating the table for composed primary keys

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

- Later on by modifying table's structure:

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

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

- Removing Primary Key

```
ALTER TABLE Students
DROP PRIMARY KEY

ALTER TABLE Students
DROP CONSTRAINT PKComposed
```

SQL Implementation of relation model (short)

- 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 UNComposed UNIQUE (StudentName, DoB, PoB) );
```

- Later on by modifying table's structure:

```
ALTER TABLE Students
ADD UNIQUE (SomeColumn)

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

- Removing Primary Key

```
ALTER TABLE Students
DROP CONSTRAINT UNComposed
```

SQL Implementation of relation model (short)

- 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 FKCourse
        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 FKComposed
        FOREIGN KEY (CourseTitle)
        REFERENCES Courses(CourseTitle)
```

- Removing Foreign Key

```
ALTER TABLE Enrollments
DROP CONSTRAINT FKComposed
```

SQL Implementation of relation model (short)

- Relational algebra is implement by SELECT

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

Projection

Selection

```
SELECT StudName, Major
FROM Students, Enrollments
WHERE Students.StudID=Enrollments.StudID
      AND CourseTitle='Databases'
```

Projection

Join (equi)

Selection

FMI

Mapping relational operators to SQL

Relational operator	SQL support
$\sigma_{\text{predicate}}(R)$	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)
$R \cup S$	R UNION S R UNION ALL S
$R - S$	R EXCEPT S R MINUS S SELECT DISTINCT R.* FROM (R LEFT OUTER JOIN S ON R.ID=S.ID) WHERE S.ID IS NULL
$R \cap S$	R INTERSECT S SELECT * FROM R (INNER | NATURAL) JOIN S
$R \times S$	SELECT * FROM R, S SELECT * FROM R CROSS JOIN S

FMI

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
$R ⟕ S$	SELECT * FROM R LEFT OUTER JOIN S ON R.commonAttrs = S.commonAttrs
$R ⟗ S$	SELECT * FROM R FULL OUTER JOIN S ON R.commonAttrs = S.commonAttrs
$R ⋉ S$	SELECT R.* FROM R NATURAL JOIN S
$R \triangleright S$	SELECT * FROM R WHERE R.commonAttr NOT IN (SELECT S.commonAttr FROM S WHERE F)
$R \div S$	
$\mathbf{S}_{AL}(R)$	SELECT <AL> FROM R
${}_{GA}\mathbf{G}_{AL}(R)$	SELECT GA1, …, GAn <AL> FROM R GROUP BY GA1, …, GAn

FMI

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. The following mnemonic can help remember the order:

Fast **W**alking **G**iants **H**ave **S**melly **O**dours

```
 FROM Clause
WHERE Clause
GROUP BY Clause
HAVING Clause
SELECT Clause
ORDER BY Clause
```

Logical Query Processing Order

To add to the above list there are two keywords used in the SELECT clauses 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.

When do we call DBMS relational?

- The $12 + 1$ Codd rules
- 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 updatable 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 types.
- 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.

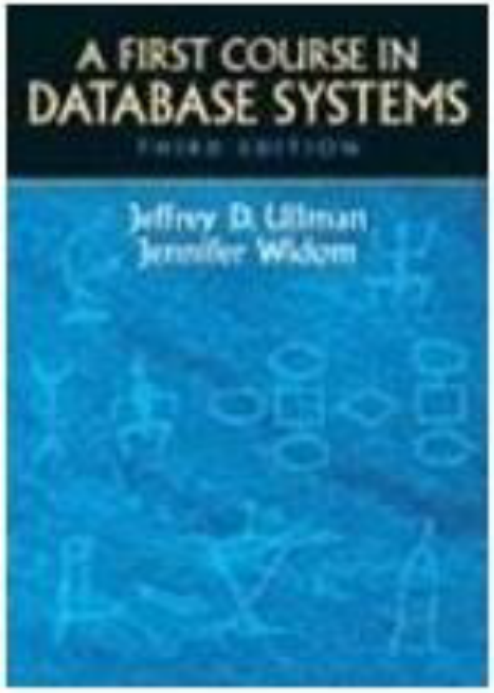
Data Manipulation rules

- 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

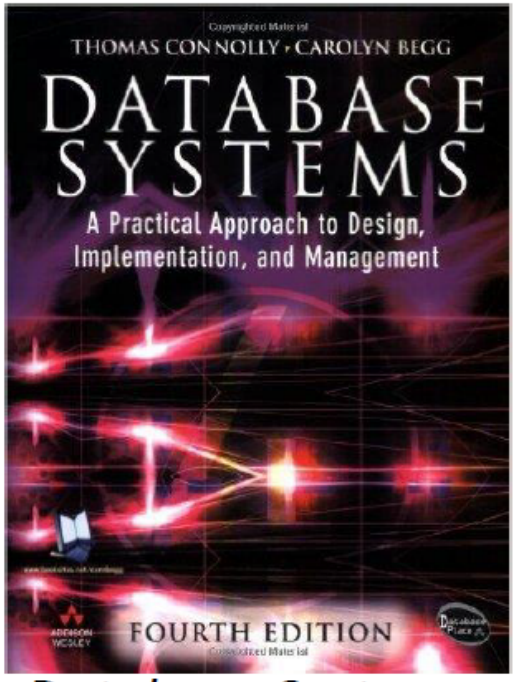
- Rule 8 (physiscal data independence): Apps remain logically unimpaired whenever any changes are made in either storage representations or access methods.
- Rule 9 (logical data independence): Apps remain logically unimpaired when information-preserving changes of any kind that permit unimpairment are 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.

Bibliography (recommended)



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

Chapter 2



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

Chapter 3 & 4

FMI