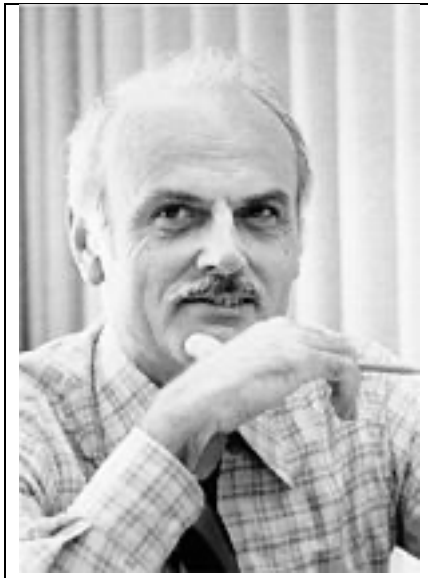
Logical data models. Relation Model. Relation Algebra.

Contents

Codd's 12 Rules



Edgar Frank "Ted" Codd
(19 August 1923 – 18 April 2003)
was an English computer scientist who, while working for IBM, invented the relational model for database management, the theoretical basis for relational databases and relational database management systems. He made other valuable contributions to computer science, but the relational model, a very influential general theory of data management, remains his most mentioned, analyzed and celebrated achievement.

The relational data model is based on the mathematical principles of **set theory and predicate logic**.
Technical article "Relational Data Model for Large Shared Databanks" by Dr. E.F. Codda, published in 1970, is the beginning of the modern theory of relational databases.

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with **twelve rules** of his own, which according to him, a database must obey in order to be regarded as a **true relational database**.

Rule 0 is a foundation rule, which acts as a base for all the other rules. Rules was added 1985. These rules can be applied on any database system that manages stored data using only its relational capabilities.

Topic about 12 Rules interpretation:

https://computing.derby.ac.uk/c/codds-twelve-rules/

Rule 0: The foundation rule:

For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a **value of some table cell**. Everything in a database must be stored in a **table format**.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a **combination of table-name, primary-key (row value), and attribute-name (column value)**. No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a **systematic and uniform** treatment. This is a very important rule because a NULL can be interpreted as one the following: data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The **structure description** of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations. This language can be used directly or by means of some application. If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

All the views of a database, which can theoretically be updated, must also be updatable by the system.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updating, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

Dr. Codd chose the term "relation" to distinguish it from the term "table", which has many different types - a table in the text, a spreadsheet, etc.

How do RDBMS work and why are they popular?

1. Users (or programs) request data from a relational database by sending it a query that is written in a special language, usually a dialect of SQL. SQL queries embed into software that provides an easier user interface.
2. In response to a query, the database returns a result set, which is just a list of rows containing the answers.
3. The simplest query is just to return all the rows from a table.
4. Often, the rows are filtered to return just the answer wanted.
5. Often, data from multiple tables are combined into one, by doing a join. Conceptually, this is done by taking all possible combinations of rows (the Cartesian product), and then filtering out everything except the answer.
6. In practice, RDBMS rewrite ("optimize") queries to perform faster, using a variety of techniques.
7. Relational operations include join, project, restrict, union, difference, intersect, and product. Then there are operators to rename columns, and summarizing or aggregating operators, and group and ungroup operators.
8. The SELECT statement in SQL serves to handle all of these except for the group and ungroup operators.
9. The flexibility of relational databases allows programmers to write queries that were not anticipated by the database designers.
10. As a result, relational databases can be used by multiple applications in ways the original designers did not foresee, which is especially important for databases that might be used for a long time (perhaps several decades).
11. This has made the idea and implementation of relational databases very popular with businesses.

Relation Data Model Concepts

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple and it has all the properties and capabilities required to process data with storage efficiency.

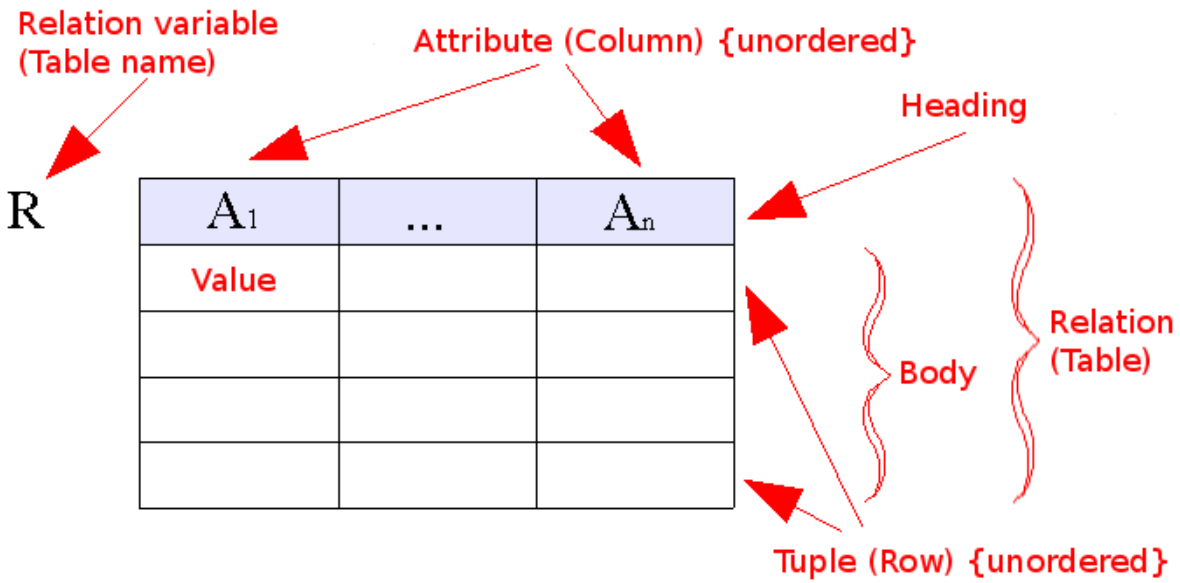
RDM Definitions

A database is composed of multiple tables (relations) and each table holds the data.

Relation (Table) − In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

Relation characteristics:

- A relation has a unique name in the database.
- No anonymous attributes (columns)
- No duplicate attribute (column) names
- No attribute (column) order significance
- No duplicate tuples (rows), because relation=set.
- No tuple (row) order significance
- Value is atomic, no repeating groups or multivalued attributes.
- Entries from columns are from the same **domain** based on their **data type** including:
  - number (numeric, integer, float, smallint,…),
  - character (string),
  - date,
  - logical (true or false), etc.
- Operations combining different data types are disallowed.



Attribute (Column) − An attribute is a named column of a relation. An attribute specification consists of its name, an indication of the data type, and a description of integrity constraints - the set of values (or domain) that this attribute can take.

Tuple (Cortege, Row) − A single row of a table, which contains a single record for that relation is called a tuple.

Domain − Every attribute has some pre-defined value scope, known as attribute domain.

A *domain* is the original sets of atomic values used to model data. By *atomic value*, we mean that each value in the domain is indivisible as far as the relational model is concerned. For example:

- The domain of Marital Status has a set of possibilities: Married, Single, Divorced.
- The domain of Shift has the set of all possible days: {Mon, Tue, Wed…}.
- The domain of Salary is the set of all floating-point numbers greater than 0 and less than 200,000.
- The domain of First Name is the set of character strings that represents names of people.

The concept of "domain" is often confused with the concept of "data type". It is necessary to distinguish between these two concepts. A data type is a **physical concept**, and a domain is a **logical concept**. For example, an integer is a data type, and age is a domain.

In summary, a domain is a set of acceptable values that a column is allowed to contain. This is based on various properties and the data type for the column.

Relation schema − describes the relation name (table name), keys, attribute names and attribute domains.

Relation key − Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Relation instance − A finite set of tuples in the relational database system represents relation instance.

Relation degree - is the number of attributes (columns) in a table.

Relation cardinality (power) - is the number of tuples (rows) in a table.

Constraints

Every relation has some conditions that must hold for it to be a valid relation. These conditions are called **Relational Integrity Constraints**. There are three main integrity constraints:

- Application constraints
- Domain constraints
- Key constraints
- Referential integrity constraints
- Database (transaction) constraints

Domain Constraints

Attributes have specific values in real-world scenario. For example, age can only be a positive integer. The same constraints have been tried to employ on the attributes of a relation. Every attribute is bound to have a specific range of values. For example, age or cost cannot be less than zero and telephone numbers cannot contain a digit outside 0-9.

Key Constraints

There must be at least one minimal subset of attributes in the relation, which can identify a tuple uniquely. This minimal subset of attributes is called **key** for that relation. If there are more than one such minimal subsets, these are called ***candidate keys***.

Key constraints force that −

- in a relation with a key attribute, no two tuples can have identical values for key attributes.
- a key attribute can not have NULL values.

Key constraints are also referred to as Entity Constraints.

Referential integrity Constraints

Referential integrity constraints work on the concept of Foreign Keys. A foreign key is a key attribute of a relation that can be referred in other relation.

Referential integrity constraint states that if a relation refers to a key attribute of a different or same relation, then that key element must exist.

Advantages and Disadvantages of Relational model

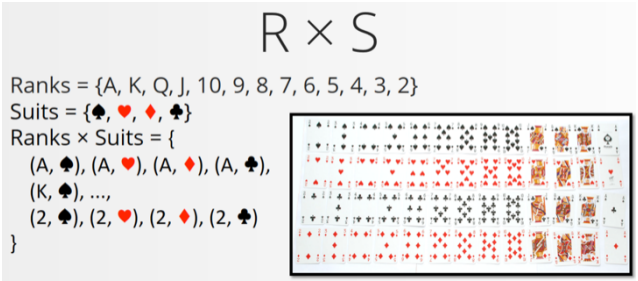
Advantages:

- **Easy to use**: The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
- **Simplicity**: A relational data model is simpler than the hierarchical and network model.
- **Theoretical basis**: the presence of a theoretical basis.
- **Query capability**: the presence of a declarative query languagelike SQL.
- **Structural Independence**: The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- **Data independence**: The structure of a database can be changed without having to change any application.
- **Scalable**: Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages:

- Few relational databases have limits on field lengths which can't be exceeded.
- Low efficiency of query execution.
- Incomplete correspondence between domain entities and relational database tables.

Relation Algebra Introduction

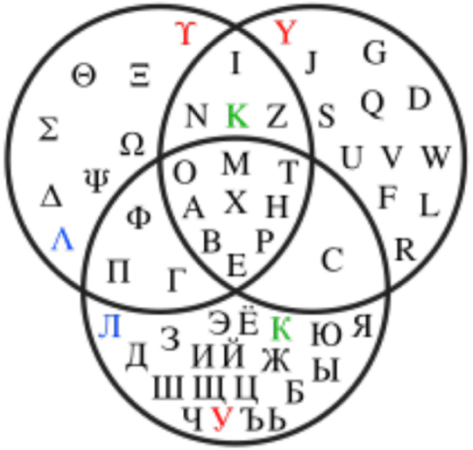


Relation data model is based on Relation algebra, and Relation algebra based on Set theory.

A **relation** is a subset of the Cartesian product of a list of **domains** characterized by a name. The steps below outline the logic between a relation and its domains.

- Domain, D - the set of values that a data item di can take.
- The Cartesian product of domains is the set of all possible combinations of domain values:
- Given n domains are denoted by D1, D2, … Dn; then (D1×D2×... ×Dn = {(d11 , …, d1i , ..., dki, …, dni)}, where dki ∈ Dk)
- And r is a relation defined on these domains. Then r ⊆ D1×D2×…×Dn
- Example 1. D1=(1,2), D2=(a,b,c). D1×D2={(1,a),(1,b),(1,c),(2,a),(2,b),(2,c)}; r1={(1,a),(1,c),(2,b)}; r2={(1,a),(1,b),(1,c),(2,a),(2,b),(2,c)}.
- Example 2. Let A=B=R, where R is the set of all real numbers. Then R×R is the set of all Cartesian coordinates of the points of the plane.

Set Theory - Definitions



Elements of a set (or points) are the objects of which the set consists. Set denoted with capital letters of the Latin alphabet, its elements are lowercase. Each element of the set is unique.

Venn diagrams are a schematic representation of all possible intersections of several sets.

Special sets

- **An empty set ∅** is a set that does not contain a single element.
- **The universe U** is a multitude containing all conceivable objects.

The cardinality of a set |A| is a characteristic of a set that generalizes the concept of the number of elements. The power of an empty set is zero; for finite sets, the power coincides with the number of elements

Relationships between sets

- **Inclusion**: A ⊆ B ⇔ ∀ a ∈ A: a ∈ B
- **Not intersection** when A and B do not have common elements: ⇔ ∀ a∈A: a ∉ B

Set Theory - Operations

Unary operations on sets:

complement: $\bar{}A:=U\setminus A=\{x|x\notin A\}$ - is the difference of the set A with the universe U.

Binary operations on sets:

intersection: $A\cap B:=\{x|x\in A\wedge x\in B\}$. If the sets A and B are not intersect, then $A\cap B = \varnothing$.

union: $A\cup B:=\{x|x\in A\vee x\in B\}$.

difference: $A\setminus B:=A\cap\bar{}B=\{x|x\in A\wedge x\notin B\}$.

symmetric difference: $A\triangle B:=(A\cup B)\setminus(A\cap B)=A\cap\bar{}B\cup\bar{}A\cap B=\{x|\ (x\in A\wedge x\notin B)\vee(x\notin A\wedge x\in B)\}$

Cartesian or direct product: $A\times B=\{(a, b)|a\in A, b\in B\}$.

Priority operations.

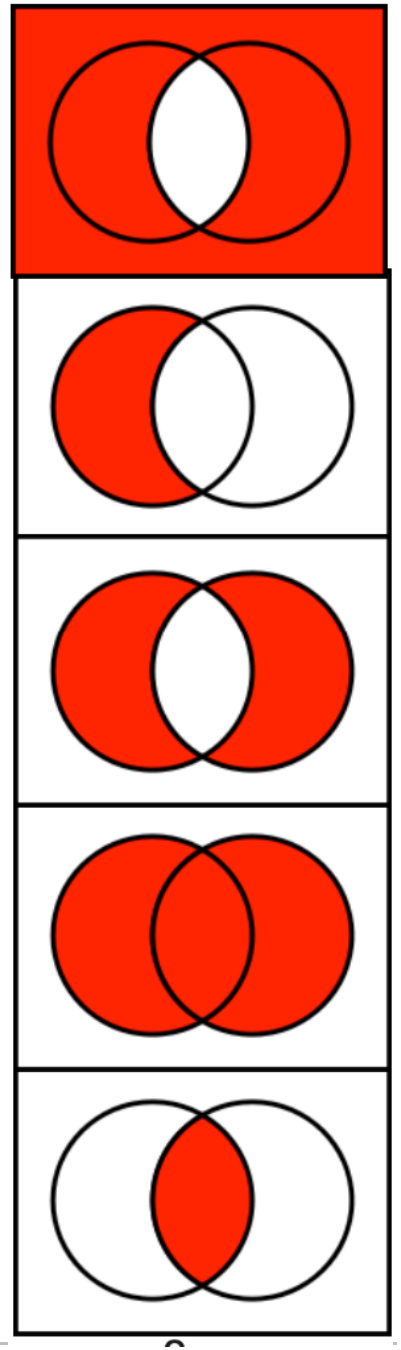
First, unary operations (addition) are performed, then intersections, then unions and differences, which have the same priority. The sequence of operations can be changed in brackets.

Set Operations Properties

1. Commutativity: $A\cup B=B\cup A$; $A\cap B=B\cap A$
2. Associativity: $(A\cup B)\cup C=A\cup(B\cup C)$; $(A\cap B)\cap C=A\cap(B\cap C)$
3. Distributivity: $(A\cup B)\cap C=(A\cap C)\cup(B\cap C)$; $(A\cap B)\cup C=(A\cup C)\cap(B\cup C)$
4. Selfie: $A\cup A=A$; $A\cap A=A$
5. Nullity: $A\cup\varnothing=\varnothing$; $A\cap\varnothing=\varnothing$
6. Duality (de Morgan laws): $\overline{A\cup B}=\bar{A}\cap\bar{B}$; $\overline{A\cap B}=\bar{A}\cup\bar{B}$

Tasks.

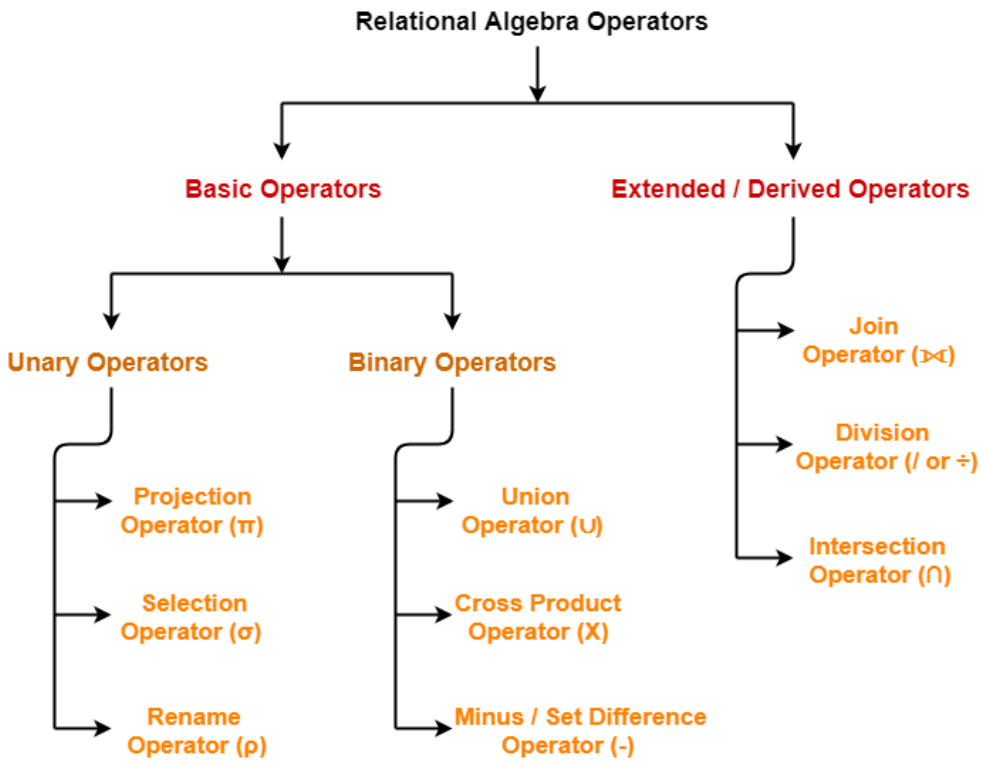
1. Find the **Venn diagram** for each given operation →.
2. How many and what other Venn diagrams can be represented for 2 sets?

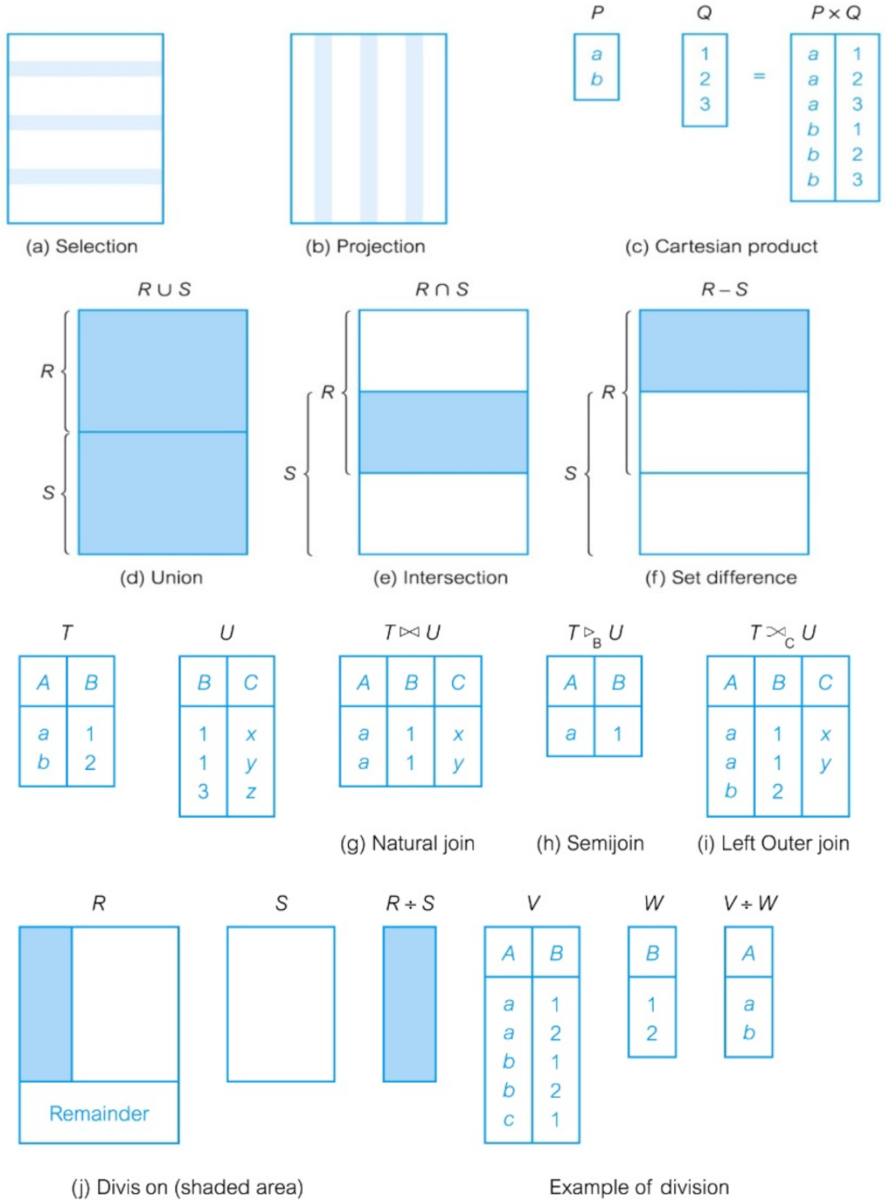


Relation Operators

Relation Operators Classification

The operators in relational algebra are classified as





Relation Operators Characteristics

Following are the important characteristics of relational operators:

- Relational Operators always work on one or more relational tables.
- Relational Operators always produce another relational table.
- The table produced by a relational operator has all the properties of a relational model.

Selection [$\sigma_{<selection\_condition>}(R)$]

This is a unary operation, the result is a subset of the original relation corresponding to the **conditions** on the values of some attributes.

Relation R

A	B	C
a	b	c
c	a	d
c	b	d

Selection $\sigma_{C=d}(R)$

A	B	C
c	a	d
c	b	d

- The number of rows returned by a selection is <= of rows in the original table.
  - Minimum Cardinality = 0
  - Maximum Cardinality = |R|
- Simple condition: AoperationB or AoperationConst, where A,B - attributes.
- We may use relational operators like =, ≠ ,> ,< ,<=, >=; logical operators like ∧ , ∨ , ! with the selection condition.
- Degree of the relation from a selection is same as input relation degree.
- Selection operator is commutative in nature:
  $\sigma_{A \wedge B}(R) \equiv \sigma_{B \wedge A}(R)$ **OR** $\sigma_B(\sigma_A(R)) \equiv \sigma_A(\sigma_B(R))$.

- Selection always selects the entire tuple. It can not select a section a tuple.
- Selection operator only selects the required tuples according to the selection condition, it does not display the selected tuples. To display the selected tuples, projection operator is used.

Projection [$\pi_{<attribute\ list>}(R)$]

This is a unary operation (performed on one relation) that serves to select a subset of attributes from the relation R.

Relation R

A	B	C
a	b	c
c	a	d
c	b	d

Projection $\pi_{A,C}(R)$

A	C
a	c
c	d

- The degree of output relation (number of columns present) is equal to the number of attributes mentioned in the attribute list.
- Projection operator automatically **removes all the duplicates** while projecting the output relation.
- If there are no duplicates in the original relation, then the cardinality will remain same otherwise it will surely reduce.
- If attribute list is a super key on R, then we will always get the same number of tuples in output relation, because no duplicates to filter.

- Projection operator does not obey commutative property: $\pi_{<list2>}(\pi_{<list1>}(R)) \neq \pi_{<list1>}(\pi_{<list2>}(R))$
- Following expressions are equivalent because both finally projects columns of list1: $\pi_{<list1>}(\pi_{<list2>}(R)) \equiv \pi_{<list1>}(R)$
- Projection does not allow duplicates while SELECT operation allows; to avoid duplicates in SQL, we use “SELECT distinct”.

Single- and different-scheme relations

- Different-scheme operation - apply to any two relationships.
- Single-scheme operation - apply to single-scheme relationships. The original relations must have the same number of columns of the some or comparable types. Comparable types are those belonging to the same data family (basic types are highlighted in bold).

Data type families (according to Oracle DBMS):		
Numeric: DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INT, INTEGER, **NUMBER,** NUMERIC, REAL, SMALLINT	**Symbolic:** CHAR, CHARACTER, LONG, LONG RAW RAW, ROWID, STRING, VARCHAR, **VARCHAR2**	**Calendare:** **DATE** TIMESTAMP

Cartesian product (cross product) [RxS]

This is a binary operation on single- or different-scheme relations, the result is a relation whose scheme includes all the attributes of the original relations. The resulting relation contains all possible combinations of tuples of the original relations.

Relation R

A	B
1	4
2	5
3	6

Relation S

C	D	E
g	h	a
a	b	c

Cartesian product RxS

A	B	C	D	E
1	4	g	h	a
1	4	a	b	c
2	5	g	h	a
2	5	a	b	c
3	6	g	h	a
3	6	a	b	c

- Cartesian product operation is both commutative and associative on relation algebra: D1XD2=D2xD1; D1x(D2xD3)=(D1xD2)xD3
- Cartesian product cardinality (tuples) =|D1xD2|=|D1|x|D2|;
- Cartesian product degree (attributes) =D1degree+D2degree

Union [RUS]

The union of two single-scheme relations R and S is the relation T=RUS, which includes all tuples of the original relations **without repetitions**.

Relation R

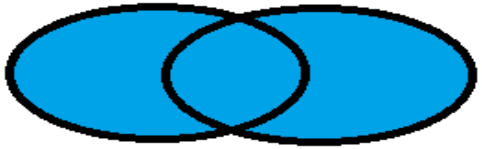
A	B	C
a	b	c
c	a	d
c	h	c

Relation S

A	B	C
g	h	a
a	b	c
h	d	d

Union RUS

A	B	C
a	b	c
c	a	d
c	h	c
g	h	a
h	d	d



- Union operation is both commutative and associative.
- In R∪S, **duplicates** are automatically **removed**.

Difference [R-S]

The difference of two single-scheme relations R and S is the set of tuples R that are not in S.

Relation R

A	B	C
a	b	c
c	a	d
c	h	c

Relation S

A	B	C
g	h	a
a	b	c
h	d	d

Difference R–S

A	B	C
c	a	d
c	h	c

Difference operation is associative but not commutative.

Intersection [R∩S]

The intersection of two single-scheme relations R and S is a subset of tuples belonging to both relations. This can be expressed in terms of the difference: R ∩ S ≡ R - (R - S). Intersection operation is both commutative and associative.

Relation R

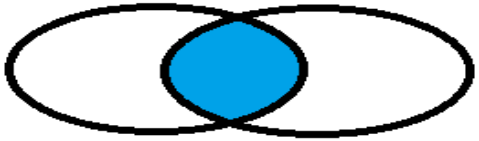
A	B	C
a	b	c
c	a	d
c	h	c
d	r	t

Relation S

A	B	C
g	h	a
a	b	c
c	a	d
g	u	v

Intersection R∩S

A	B	C
a	b	c
c	a	d



Division [R/S]]

- Division is the inverse of Cartesian product.
- Relations R and S two different-scheme, but they must have some attributes in common.
- Let the R contains attributes {r1,r2,...,rk,rk+1,...,rn}, and the S contains {rk+1,...,rn}, then relation R/S contains the attributes {r1, r2, ..., rk}.
- The division can be expressed as follows: $R/S \equiv \pi_{r1,...,rk}(R) - \pi_{r1,...,rk}((\pi_{r1,...,rk}(R) \times S) - R)$.

Relation R

A	B	C	D
a	b	c	b
a	b	g	h
c	f	g	h
c	f	c	b
a	v	c	b
c	v	g	h

Relation S

C	D
c	b
g	h

Division R/S

A	B
a	b
c	f

In simple arithmetic Division is the inverse of Multiple, if:

- Value S = 3
- Value T = 2
- Value R = 6

then: R = S * T (i.e. 6 = 2*3)
and: R / T = S (i.e. 6/2 = 3)
and: R / S = T (i.e. 6/3 = 2)

Join [R>_FS]

- Join (or general join or **θ-join**) defines a **subset** (party) of the Cartesian product of two different-scheme relation. A tuple of a Cartesian product is included in the resulting relation if some condition F is executed for the attributes of different initial relations.
- We may use relational operators like =, ≠ ,> ,< ,<=, >= with the selection condition F.
- A join can be expressed as follows: $R\mathbf{><_F}S \equiv \sigma_F(RxS)$
- If the condition is the equivalence (=) of the values of two attributes of the original relations, such an operation is called an **equi-join**.
- Equi-join is called **natural join** by the same attributes of the original relations.

Relation R

A	B	C
a	b	c
c	a	d
c	h	c
g	b	d

Relation S

A	D	E
g	h	a
c	b	c
h	d	d

Natural Join R><S

A	B	C	D	E
c	a	d	b	c
c	h	c	b	c
g	b	d	h	a

Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

Relation algebra and SQL

	Name	Relational Algebra	SQL
Basic / Complete Set	**select**	$\sigma_{age>18}(patients)$	`select * from patients where age > 18`
	project	$\pi_{patient\_id,name}(patients)$	`select patient_id, name from patients`
	product	patients $\times$ medical_records	`select * from patients, medical_records`

	union
	(note: union tables must have the same number of columns and same data types)
		$\pi_{name}(patients) \cup \pi_{name}(doctors)$	`select name from patients`
			`union`
			`select name from doctors`

	difference
	(second table is not necessarily a subset of the first)
		$\pi_{name}(patients) - \pi_{name}(doctors)$	`select name from patients`
			`minus`
			`select name from doctors`

Derived	**rename**	$\rho_{staff}(\pi_{patient\_id,name}(patients))$	`select * from (`
			`  select patient_id, name`
			`  from patients`
			`) as staff`

	intersection
	note: A intersect B = A-(A-B)
		$\pi_{name}(patients) \cap \pi_{name}(doctors)$	`select name from patients`
			`intersect`
			`select name from doctors`

	natural join note: lecture notes use star, but every other source seems to use bowtie	patients $\bowtie$ medical_records or patients $*$ medical_records	`select * from patients` `natural join medical_records`

	theta join
	note: you may sometimes see the '=' replaced with 'θ'
		patients $\bowtie_{patients.id=doctors.patient\_id}$ (doctors)	`select * from patients`
			`join doctors`
			`on patients.id=doctors.patient_id`

Query Optimization using Relational Algebra

Problem of large query.

Since the degree and cardinality in the initial relations can be large (tens, hundreds), it is advisable to first apply selection and projection, and only then form the Cartesian product.

Example.

So, if two relations have n tuples each and the access time to each record is t0, then the total memory access time to form the full Cartesian product is:

$$\mathbf{T_{access} = n^2 t_0}.$$

If $n = 10^4$, $t_0 = 10$ ms, then

$$T\ access = 10^4 x 10^4 10^{-2} = 10^6 \text{ sec} = 11{,}5 \text{ days}.$$

Therefore, in order to **save machine time**, it is necessary to perform preliminary optimization of queries to the relational database.

Query Optimization Idea

The idea is to search a space of equivalent algebraic expressions, estimating the run time of each, and trying to **find the minimum**.

The general optimization strategy is as follows:

- perform selection and projection as early as possible before Cartesian multiplication (in order to reduce arity and the number of tuples);
- collect cascades of selection and projection in order to perform them in one file view;
- process (sort, index) files before making a connection;
- combine projections with previous or subsequent two-seater operations.

To implement this strategy, **equivalent expressions** of relational algebra are used.

Some equivalences for optimization:

1. **commutativity of select**, which means we can select rows in either order of two conditions:

   $\sigma_{\text{cond A}}(\sigma_{\text{cond B}}(R)) \equiv \sigma_{\text{cond B}}(\sigma_{\text{cond A}}(R))$

2. **cascading of select,** which means we can split up a conjunctive condition into multiple steps:

   $\sigma_{\text{cond A} \wedge \text{B}}(R) \equiv \sigma_{\text{cond A}}(\sigma_{\text{cond B}}(R))$

3. **cascading of projections,** which means we can project out unwanted columns early. Note that $X \subseteq Y$ and both are a list of column names.

   $\pi_X(R) \equiv \pi_X(\pi_Y(R))$

4. cross products and joins are **commutative**.

   $R \times S \equiv S \times R$

5. cross products and joins are **associative**:

   $R \times (S \times T) \equiv (S \times R) \times T$

6. selection and projection can be **commutative** if the relevant attributes are retained:

   $\pi_X(\sigma_{\text{cond A}}(R)) \equiv \sigma_{\text{cond A}}(\pi_X(R))$

7. selection and cross-product (or join) can be **commutative** if the selection condition only applies to attributes of one relation.

   $\sigma_{\text{cond A}}(R \times S) \equiv \sigma_{\text{cond A}}(R) \times S$

8. projection is **commutative** with cross product and join:

   $\pi_X(R \times S) \equiv \pi_X(R) \times \pi_X(S)$

9. And others.

The Merge-sort Idea

For cost estimation big-O notation usage, plus number of tuples in different relations and results of sub-queries.

- If you think a select will be very small and the other relation has an index, do the selection first and then look up the remaining tuples using the index.
- If you think a select will be very big, and both relations are sorted, do the join using the merge-sort idea and ignore the index.

To find the intersection or union or difference of two sets, you need to make sure there are no duplicates. In general, we are merging the sets. You can, of course, merge them and remove duplicates afterwards, but it is often faster to remove them as we merge.

One of the most efficient ways to remove duplicates is to sort, so that identical elements end up next to each other. Sorting is O(N log N), and most any other algorithm will end up being O(N2), as you iterate through one set, looking in the other.

The merge-sort algorithm has two sorted subsets, and merges them in O(N) time to yield a sorted merger. We can use that idea to produce the union of the two subset, the intersection, and so forth.