UNIT – II

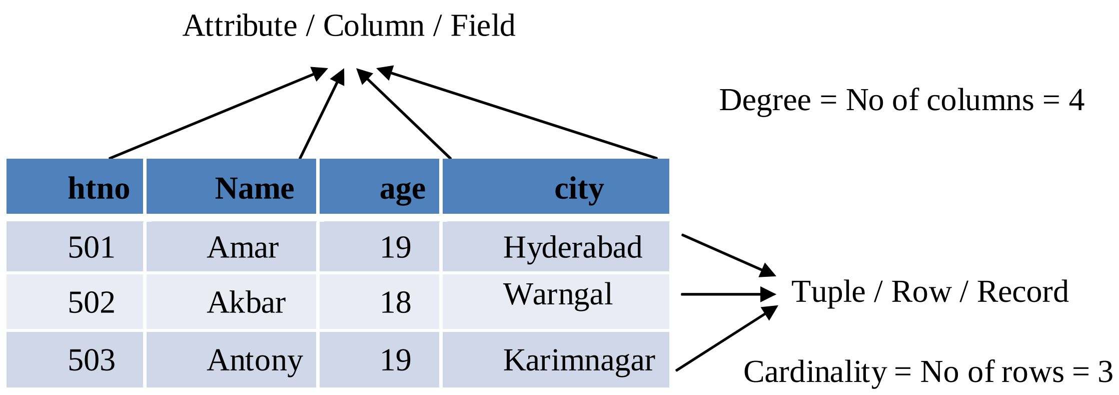
Introduction to the Relational Model: Integrity constraint over relations, enforcing integrity constraints, querying relational data, logical data base design, introduction to views, destroying/altering tables and views. Relational Algebra, Tuple relational Calculus, Domain relational calculus.

1. RELATIONAL MODEL

Relational data model is the most popular data model used widely around the world for data storage. In this model data is stored in the form of tables.

Relational Model Concepts

Table is also called Relation. Let the below table name be SUDENT_DATA


Attribute / Column / Field
Degree = No of columns = 4
Tuple / Row / Record
Cardinality = No of rows = 3

htno	Name	age	city
501	Amar	19	Hyderabad
502	Akbar	18	Warngal
503	Antony	19	Karimnagar

Table: In relational model the data is saved in the form of tables. A table has two properties rows and columns. Rows represent records and columns represent attributes.

Attribute: Each column in a Table is an attribute. Attributes are the properties that define a relation. e.g., HTNO, NAME, AGE, CITY in the above relation.

Tuple: Every single row of a table is called record or tuple.

Relation Schema: It represents the name of the relation (Table) with its attributes. Eg., STUDENT_DATA(htno, name, age, city)

Degree: The total number of attributes in the relation is called the degree of the relation.

Cardinality: Total number of rows present in the Table.

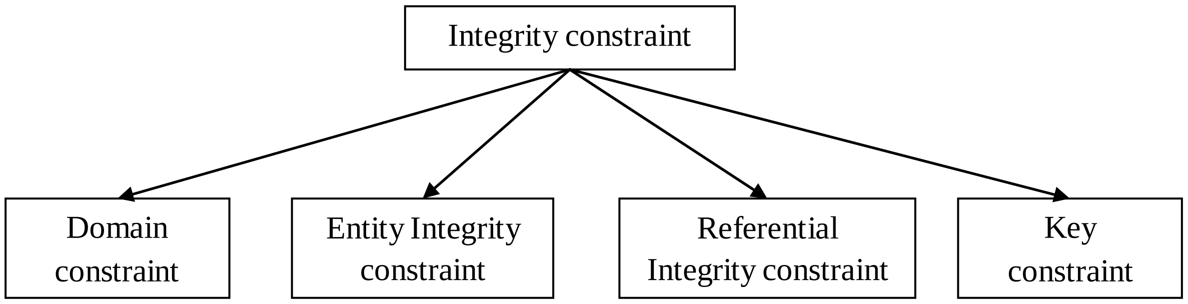
2. INTEGRITY CONSTRAINT

- Integrity constraints are a set of rules that the database should not violate.
- Integrity constraints ensure that authorized changes (update deletion, insertion) made to the database should not affect data consistency.
- Integrity constraints may apply to attribute or to relationships between tables.

TYPES OF INTEGRITY CONSTRAINTS

The integrity constraints supported by DBMS are:

1. Domain Integrity Constraint
2. Entity Integrity Constraint
3. Referential Integrity Constraint
4. Key Constraints



➢ **Domain Constraint:** These are attribute level constraints. An attribute can only take values which lie inside the domain range. **Example:** If a constrain AGE > 0 is applied on STUDENT relation, inserting negative value of AGE will result in failure. If the domain of AGE is defined as *integer*, inserting an alphabet in age column is not accepted.

Example:

ID	**NAME**	**SEMESTER**	**AGE**
1001	TOM	I	18
1002	JHONSON	IV	20
1003	KATE	VI	21
1004	JHON	II	19
1005	MORGAN	II	A

Not allowed. Because AGE is an integer attribute

➢ **Entity integrity constraints:** The entity integrity constraint states that primary key value can't be null. This is because the primary key value is used to identify individual rows in relation. A table can contain a null value other than the primary key field.

Example: Let ID be the primary key in the below table.

ID	NAME	SEMESTER	AGE
1001	TOM	I	18
1002	JHONSON	IV	20
	KATE	VI	21

Not allowed. Because primary key can't be NULL value.

- **Referential Integrity Constraints:** It is also called as foreign key constraint. A referential integrity constraint is specified between two tables. In this type of constraints, if a foreign key in Table 2 refers to the Primary Key of Table 1, then every value of the Foreign Key in Table 2 must be null or be available in Table 1.

  Example: *Department Table* **(Table 1)**

  Primary Key

Dept_No	Dept_Name
05	CSE
02	EEE
04	ECE

  Primary Key

  Relationship

  Employee Table **(Table 2)**

EID	NAME	AGE	Dept_No
1001	TOM	45	04
1002	JHONSON	38	01
1003	KATE	54	05
1004	MORGAN	29	02

  Foreign Key

  Not allowed as Dept_No 01. Because 01 value is not present as a primary key in Table1. Dept_No is a foreign key defined in Table2.

- **Key Constraints:** A Key Constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple. There are 4 types of key constraints. They are

  i. **Candidate key:** The candidate keys in a table are defined as the set of keys that is minimal and can uniquely identify any data row in the table.
  ii. **Primary key:** It can uniquely identify any data row of the table. The primary key is one of the selected candidate key.
  iii. **Super key:** Super Key is the superset of primary key. The super key contains a set of attributes, including the primary key, which can uniquely identify any data row in the table.

iv. **Foreign key:** It is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

Composite Key: If any single attribute of a table is not capable of being the key i.e it cannot identify a row uniquely, then we combine two or more attributes to form a key. This is known as a composite key.

Secondary Key: Only one of the candidate keys is selected as the primary key. The rest of them are known as secondary keys.

3. ENFORCING INTEGRITY CONSTRAINTS

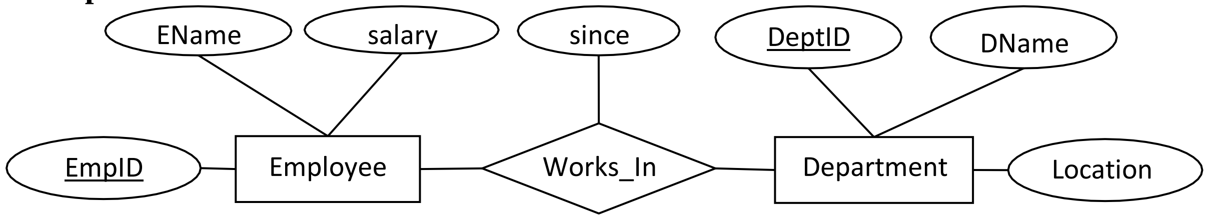
Database Constraints are declarative integrity rules of defining table structures. They include the following 7 constraint types:

1. **Data type constraint:** This defines the type of data, data length, and a few other attributes which are specifically associated with the type of data in a column.
2. **Default constraint:** This defines what value the column should use when no value has been supplied explicitly when inserting a record in the table.
3. **Nullability constraint:** This defines that if a column is NOT NULL or allow NULL values to be stored in it.
4. **Primary key constraint:** This is the unique identifier of the table. Each row must have a distinct value. The primary key can be either a sequentially incremented integer number or a natural selection of data that represents what is happening in the real world (e.g. Social Security Number). NULL values are not allowed in primary key values.
5. **Unique constraint:** This defines that the values in a column must be unique and no duplicates should be stored. Sometimes the data in a column must be unique even though the column does not act as Primary Key of the table. Only one of the values can be NULL.
6. **Foreign key constraint:** This defines how referential integrity is enforced between two tables.
7. **Check constraint:** This defines a validation rule for the data values in a column so it is a user-defined data integrity constraint. This rule is defined by the user when designing the column in a table.

4. LOGICAL DATABASE DESIGN

1. Each entity in the ER model will become a table and all attributes of that entity will become columns of the table. Key attribute of the entity will become primary key in the table.

Example:



Employee

EmpID	EName	salary

Department

DeptID	DName	Location

```
CREATE TABLE Employee
 (
      EmpID NUMBER(3),
      EName VARCHAR(20),
      Salary    NUMBER(5),
      PRIMARY KEY(EmpID)
);
```

```
CREATE TABLE Department
 (
      DeptID NUMBER(3),
      DName VARCHAR(15),
      Location VARCHAR(15),
      PRIMARY KEY(DeptID)
);
```

2. Each relationship in the ER model will become a table. Key attributes of participating entities in the relationship will become columns of the table. If the relationship has any attributes, then they also will become columns of the table.

Example: From the above ER diagram, the Works_In relationship converted as

Employee

EmpID	EName	Salary

Department

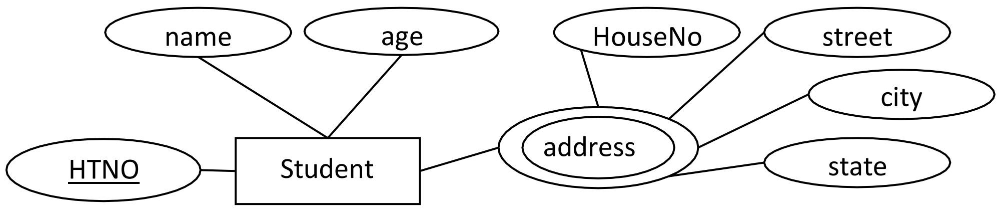
DeptID	DName	Location

Works_In

EmpID	DeptID	since

```
CREATE TABLE Works_In
 (
      EmpID NUMBER(3),
      DeptID NUMBER(3),
      Since       DATE,
      PRIMARY KEY(EmpID, DeptID),
      FOREIGN KEY (EmpID) REFERENCES Employee(EmpID),
      FOREIGN KEY (DeptID) REFERENCES Department(DeptID),
);
```

3. Any multi-valued attribute is converted into new table. The primary key of the entity will be added as column in the new table.



Student

HTNO	name	age

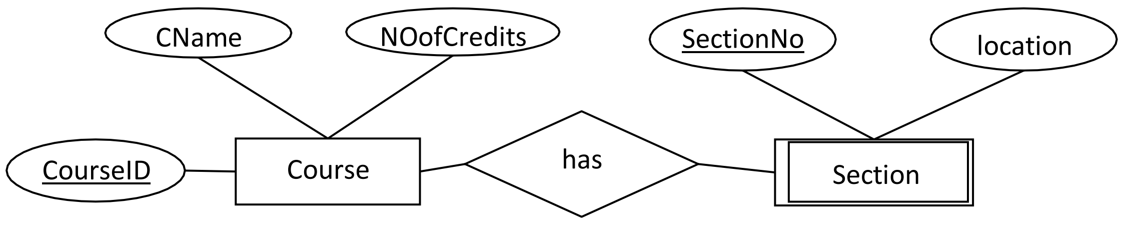
Address

HTNO	houseNo	street	city	state

```
CREATE TABLE Student
 (
      HTNO   CHAR(10),
      name    VARCHAR(20),
      age NUMBER(2),
      PRIMARY KEY(HTNO)
);
```

```
CREATE TABLE Address
 (
      HTNO  CHAR(10),
      houseNo NUMBER(3),
      street    VARCHAR(20),
      city  VARCHAR(15),
      state    VARCHAR(15),
      PRIMARY KEY(HTNO),
      FOREIGN KEY (HTNO) REFERENCES Student(HTNO),
);
```

4. Each weak entity is converted into a table with all its attributes as columns and primary key of the strong entity acts as a foreign key in this table.



Course

CourseID	CName	NOofCredits

Section

CourseID	SectionNo	location

```
CREATE TABLE Course
 (
      CourseID NUMBER(2),
      CName    VARCHAR(20),
      NOofCredits NUMBER(2),
      PRIMARY KEY(CourseID)
);
```

```
CREATE TABLE Section
 (
   SectionNo CHAR(2),
   CourseID NUMBER(2),
   location    VARCHAR(15),
  PRIMARY KEY(CourseID, SectionNo),
 FOREIGN KEY(CourseID) REFERENCES Course(CourseId)
);
```

5. INTRODUCTION TO VIEWS

A view is virtual tables whose rows are not explicitly stored in the database but are computed as needed from a view definition. They are used to restrict access to the database or to hide data complexity. A view contains rows and columns, just like a real table. Creating a view does not take any storage space as only the view query is stored in the data dictionary and the actual data is not stored. The tables referred in the views are known as Base tables. Views do not contain data of their own. They take data from the base tables.

The reasons for using views are

- Security is increased - sensitive information can be excluded from a view.
- Views can represent a subset of the data contained in a table.
- Views can join and simplify multiple tables into a single virtual table.
- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data it presents.
- Different views can be created on the same base table for different categories of users.

Creating Views syntax:

```
CREATE VIEW view_name  AS
SELECT column_list
FROM table_name [WHERE condition] ;
```

Examples: Consider the below given employees table. employees(eid, name, salary, experience)

employees

eid	ename	salary	Experience
101	Jhon	20000	2
105	Sam	18000	2
108	Ram	30000	4

If we want to hide the salary column from accessing a group of users, then we can create view on employees table as follows.

```
CREATE VIEW  emp AS
SELECT eid, ename, experience FROM employees;
```

emp

eid	Name	Experience
101	Jhon	2
105	Sam	2
108	Ram	4

The view *emp* is a virtual table. The data in the *emp* table is not saved in the database but collected from *employees* table whenever *emp* table is referred in SQL query. We can perform all operations (INSERT, DELETE, UPDATE) on a view just like on a table but under some restrictions.

When can insertion, delete or update performed on view?

- The view is defined from one and only one table.
- The view must include the PRIMARY KEY of the base table.
- The base table columns which are not part of view should not have NOT NULL constraint.
- The view should not have any field made out of aggregate functions.
- The view must not have any DISTINCT clause in its definition.
- The view must not have any GROUP BY or HAVING clause in its definition.
- The view must not have any SUBQUERIES in its definitions.

i. **Inserting Rows into a View:** A new row can be inserted into a view in a similar way as you insert them in a table. When an insert operation performed on view, first a new row is inserted into the base table and the same is reflected in the view.

ii. **Deleting Rows into a View:** A row(s) can be deleted from a view in a similar way as you delete them from a table. When an delete operation performed on view, first row(s) is/are deleted from the base table and the same is reflected in the view.

iii. **Updating Rows into a View:** A row(s) can be updated in a view in a similar way as you update them in a table. When an update operation performed on view, first data is updated in the base table and the same is reflected in the view.

iv. **Dropping/Destroying View:** Whenever you do not need the view anymore, we can destroy the view by using DROP command. The syntax is very simple and is given below −

```
DROP VIEW view_name;
```

Example: DROP VIEW emp;

6. RELATIONAL ALGEBRA

Relational Algebra is procedural query language, which takes Relation as input and generates relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.

Operator Symbol	Operator Name	Explanation
π	Projection	Select column names
σ	Selection	Select row values
ρ	Renaming	Rename a table name or expression results
$\cup$	Union	Perform union operation
$\cap$	Intersection	Perform intersection operation
-	Set deference	Perform set difference operation
$\times$	Cartesian product	Every row of first table is joined with every row of second table
$\bowtie$	Join	Join two tables based on some condition

i. **Select Operation (σ):** It selects tuples that satisfy the given predicate from a relation.
Notation : $\sigma_p(r)$

where σ stands for selecting tuples (rows) and r stands for relation (table) name. *p* is prepositional logic formula which may use connectors like ***and, or***, and ***not***. These terms may use relational operators like =, ≠, ≥, < , >, ≤.

Example 1: $\sigma_{\texttt{subject = "database"}}$(Books)
Output : Selects rows whose subject is 'database' from books table.

Example 2: $\sigma_{\texttt{subject = "database" and price = "450"}}$(Books)
Output : Selects rows from books where subject is 'database' and 'price' is 450.

Example 3: $\sigma_{\texttt{subject = "database" and price = "450" or year > "2010"}}$(Books)
Output : Selects rows from books where subject is 'database' and 'price' is 450 or those books published after 2010.

ii. **Project Operation (∏):** It projects column(s) that satisfy a given predicate.
Notation: $\prod_{A1, A2, \ldots An}$ (r)

where A_1, A_2 , A_n are column (attribute) names of relation **r**. Duplicate rows are automatically eliminated in the output.

Example: $\prod_{\text{subject, author}}$ (Books)

Display values from columns subject and author from the relation Books.

iii. **Union Operation (⊠):** It performs union operation between two given relations. It combines rows from two given relations.

Notation: r U s

Where **r** and **s** are either database relations or relation result set (temporary relation). r U **s** returns a relation instance containing all tuples that occur in either relation instance r or relation instance **s** (or both). For a union operation to be valid, the following conditions must hold:

- **r** and **s** must have the same number of attributes.
- Attribute domains must be compatible in **r** and **s**.

Example: $\prod_{author}$(Books) ∪ $\prod_{author}$(Articles)

Output: Projects the names of the authors who have either written a book or an article or both.

iv. **Intersection Operation (∩):** It performs intersection operation between two given relations . It collect only rows which are common in the two given relations.

Notation: R ∩ S

R ∩ *S* returns a relation instance containing all tuples that occur in *both R* and *S*. The relations *R* and *S* must be union-compatible, and the schema of the result is defined to be identical to the schema of *R*.

$\prod_{author}$(Books) ∩ $\prod_{author}$(Articles)

Output: Projects the names of the authors who have written both book and an article.

v. **Set Difference (−):** It finds tuples which are present in one relation but not in the second relation.

Notation: r − **s**

Finds all the tuples that are present in **r** but not in **s**.

Example: $\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Output − Provides the name of authors who have written books but not articles.

vi. **Cartesian Product (X):** It returns a relation instance whose schema contains all the fields of table-1 (in the same order as they appear in table-1) followed by all the fields of table-2. It combines every row in first table with every row in the second table.

Notation: r X **s**

Where **r** and **s** are relations and their output will be defined as : r X s = { q t | q ∈ r and t ∈ s}

vii. **Natural join (⋈):**The most general version of the join operation accepts a *join condition* c and
a pair of relation instances as arguments and returns a relation instance. The *join condition* is identical to a *selection condition* in form. The operation is defined as follows:

$R \bowtie_c S = \sigma_c(R \times S)$

Thus $\bowtie$ is defined to be a cross-product followed by a selection. Note that the condition c can refer to attributes of both Rand *S*.

Note: *If the condition c in R* $\bowtie_c$ *S contain equal operator, then it is called equi-join*

viii. **Natural Join($\bowtie$):** In this case, we can simply omit the join condition; the default is that the join condition is a collection of equalities on all common fields. We call this special case as natural join, and it has the nice property that the result is guaranteed not to have two fields with the same name.

ix. **Rename Operation (ρ):** The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho ρ.

Notation: ρ(temp, E)

Where the result of expression **E** is saved with name of ***temp***.

x. **Division (/):** Consider two relation instances *A* and *B* in which *A* has (exactly) two fields *x* and *y* and *B* has just one field *y,* with the same domain as in *A.* We define the *division* operation *A / B* as the set of all *x* values (in the form of unary tuples) such that for *every y* value in (a tuple of) *B,* there is a tuple *(x,y)* in *A.*

Example:

A

SNO	PNO
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

B1

PNO
P2

B2

PNO
P2
P4

B3

PNO
P2
P4

A / B1

SNO
S1
S2
S3
S4

A / B2

SNO
S1
S4

A / B3

SNO
S1

Sample Queries: We present a number of sample queries using the following schema:

Sailors (*sid:* integer, *sname:* string, *rating:* integer, *age:* real) Boats (*bid:* integer, *bname:* string, *color:* string)

Reserves (*sid:* integer, *bid:* integer, *day:* date)

The key fields are underlined, and the domain of each field is listed after the field name. Thus *sid* is the key for Sailors, *bid* is the key for Boats, and all three fields together form the key for Reserves. Fields in an instance of one of these relations will be referred to by name, or positionally, using the order in which they are listed above.

(Q1) Find the names of sailors who have reserved boat 103.

This query can be written as follows:

$\pi_{sname}((\sigma_{bid=103} Reserves) \bowtie Sailors)$

We first compute the set of tuples in Reserves with *bid* = 103 and then take the natural join of this set with Sailors. This expression can be evaluated on instances of Reserves and Sailors. Evaluated on the instances *R*2 and *S*3, it yields a relation

(Q2) Find the names of sailors who have reserved a red boat.

$\pi_{sname}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors$

This query involves a series of two joins. First we choose (tuples describing) red boats. Then, we join this set with Reserves (natural join, with equality specified on the **bid** column) to identify reservations of red boats. Next, we join the resulting intermediate relation with Sailors (natural join, with equality specified on the **sid** column) to retrieve the names of sailors who have rnade reservations for red boats. Finally, we project the sailors' names.

(Q3) Find the colors of boats reserved by Lubber.

$\pi_{color}((\sigma_{sname='Lubber'} Sailors) \bowtie Reserves \bowtie Boats)$

(Q4) Find the names of sailors who have reserved at least one boat.

$\pi_{sname}(Sailors \bowtie Reserves)$

The join of Sailors and Reserves creates an intermediate relation in which tuples consist of a Sailors tuple 'attached to' a Reserves tuple. A Sailors tuple appears in (some tuple of) this intermediate relation only if at least one Reserves tuple has the same sid value, that is, the sailor has made some reservation.

(Q5) Find the names of sailors who have reserved a red or a green boat.

$\rho(Tempboats, (\sigma_{color='red'} Boats) \cup (\sigma_{color='green'} Boats))$

$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$

We identify the set of all the rows that are either red or green from boats table. We rename this result as Tempboats. Then we join Tempboats with Reserves to identify sid's of sailors. Finally, we join with Sailors to find the names of Sailors with those sids.

(Q6) Find the names of sailors who have reserved a red and a green boat

$$\rho(Tempboats2, (\sigma_{color='red'} Boats) \cap (\sigma_{color='green'} Boats))$$

$$\pi_{sname}(Tempboats2 \bowtie Reserves \bowtie Sailors)$$

However, this solution is incorrect-it instead tries to compute sailors who have reserved a boat that is both red and green. A boat can be only one color; this query will always return an empty answer set. The right answer is

$$\rho(Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves))$$

$$\rho(Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie Reserves))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

The two temporary relations compute the sids of sailors, and their intersection identifies sailors who have reserved both red and green boats.

(Q7) Find the names of sailors who have reserved at least two boats.

$$\rho(Reservations, \pi_{sid,sname,bid}(Sailors \bowtie Reserves))$$

$$\rho(Reservationpairs(1 \rightarrow sid1, 2 \rightarrow sname1, 3 \rightarrow bid1, 4 \rightarrow$$

$$sid2, 5 \rightarrow sname2, 6 \rightarrow bid2), Reservations \times Reservations)$$

$$\pi_{sname1}\sigma_{(sid1=sid2) \cap (bid1=bid2)} Reservationpairs$$

First, we compute tuples of the form (sid, sname, bid), where sailor sid has made a reservation for boat bid; this set of tuples is the temporary relation Reservations. Next we find all pairs of Reservations tuples where the same sailor has made both reservations and the boats involved are distinct. Here is the central idea: To show that a sailor has reserved two boats, we must find two Reservations tuples involving the same sailor but distinct boats. Finally, we project the names of such sailors.

(Q8) Find the sids of sailors with age over 20 who have not reserved a red boat.

$$\pi_{sid}(\sigma_{age>20} Sailors) - \pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves \bowtie Sailors)$$

This query illustrates the use of the set-difference operator. Again, we use the fact that *sid* is the key for Sailors. We first identify sailors aged over 20 instances and then discard those who have reserved a red boat to obtain the answer.

(Q9) Find the names of sailors who have reserved all boats.

The use of the word *all* (or *every*) is a good indication that the division operation might be applicable:

$$\rho(Tempsids, (\pi_{sid,bid}Reserves)/(\pi_{bid}Boats))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

***(Q10) Find the names of sailors who have reserved all boats called Interlake*.**

$$\rho(Tempsids, (\pi_{sid,bid}\ Reserves)/(\pi_{bid}(\sigma_{bname='Interlake'}\ Boats)))$$

$$\pi_{sname}(Tempsids \bowtie Sailors)$$

7. RELATIONAL CALCULUS

Relational calculus is an alternative to relational algebra. In contrast to the algebra, which is procedural, the calculus is nonprocedural, or *declarative*, in that it allows us to describe the set of answers without being explicit about how they should be computed.

Tuple Relational Calculus

Tuple Relational Calculus is a **non-procedural query language** unlike relational algebra. Tuple Calculus provides only the description of the query but it does not provide the methods to solve it. Thus, it explains what to do but not how to do.

In Tuple Relational Calculus, a query is expressed as `{t| P(t)}`

where t = resulting tuples, P(t) = known as Predicate and these are the conditions that are used to fetch t. Thus, it generates set of all tuples t, such that Predicate P(t) is true for t.

P(t) may have various conditions logically combined with OR (∨), AND (∧), NOT(¬).

It also uses quantifiers:

∃ t ∈ r (Q(t)) = ”there exists” a tuple in t in relation r such that predicate Q(t) is true.

∀ t ∈ r (Q(t)) = Q(t) is true “for all” tuples in relation r.

(Q12) Find the names and ages of sailors with a rating above 7 .

$$\{\, P \mid \exists S \in Sailors(S.rating > 7 \wedge P.name = S.sname \wedge P.age = S.age)\}$$

This query illustrates a useful convention: P is considered to be a tuple variable with exactly two fields, which are called *name* and *age*,.

(Q13) Find the sailor name, boat id, and reservation date for each reservation

$$\{P \mid \exists R \in Reserves \quad \exists S \in Sailors$$

$$(R.sid = S.sid \wedge P.bid = R.bid \wedge P.day = R.day \wedge P.sname = S.sname)\}$$

(Q1) Find the names of sailors who have reserved boat 103*. (similar question Q1 from relational algebra)*

$$\{P \mid \exists S \in Sailors\ \exists R \in Reserves(R.sid = S.sid \wedge R.bid = 103 \wedge P.sname \wedge S.sname)\}$$

This query can be read as follows: “Retrieve all sailor tuples for which there exists a tuple in Reserves, having the same value in the *sid* field, and with *bid* = 103.”

(Q2) Find the names of sailors who have reserved a red boat. *(similar question Q2 from relational algebra)*

$$\{P \mid \exists S \in Sailors\ \exists R \in Reserves(R.sid = S.sid \wedge P.sname = S.sname$$

$$\wedge \exists B \in Boats(B.bid = R.bid \wedge B.color = 'red'))\}$$

This query can be read as follows: “Retrieve all sailor tuples S for which there exist tuples R in Reserves and B in Boats such that $S.sid = R.sid$, $R.bid = B.bid$, and $B.color = 'red'$.”

(Q7) Find the names of sailors who have reserved at least two boats. *(similar question Q7 from relational algebra)*

$$\{P \mid \exists\ S \in Sailors\ \exists R1 \in Reserves\ \exists R2 \in Reserves\ (S.sid = R1.sid$$
$$\wedge R1.sid = R2.sid \wedge R1.bid \neq R2.bid \wedge P.sname = S.sname)\}$$

(Q9) Find the names of sailors who have reserved all boats. *(similar question Q9 from relational algebra)*

$$\{P \mid \exists\ S \in Sailors \quad \forall B \in Boats$$

$$(\exists\ R \in Reserves(S.sid = R.sid \wedge R.bid = B.bid \wedge P.sname = S.sname))\}$$

(Q14) Find sailors who have reserved all red boats.

$$\{S \mid S \exists\ Sailors \in\ \forall B \in\ Boats$$

$$(B.color = 'red' \Rightarrow (\exists R \in Reserves(S.sid = R.sid \wedge R.bid = B.bid)))\}$$

Domain Relational Calculus

A domain variable is a variable that ranges over the values in the domain of some attribute (e.g., the variable can be assigned an integer if it appears in an attribute whose domain is the set of integers).

A DRC query has the form $\{ \langle x_1, x_2, \ldots, x_n \rangle \mid p(\langle x_1, x_2, \ldots, x_n \rangle)\}$

where each x_i is either a *domain variable* or a constant and $p(\langle x_1, x_2, \ldots, x_n \rangle)$ denotes a DRC formula whose only free variables are the variables among the x_i, $1 \le i \le n$. The result of this query is the set of all tuples $\langle x_1, x_2, \ldots, x_n \rangle$ for which the formula evaluates to true.

A DRC formula is defined in a manner very similar to the definition of a TRC formula. The main difference is that the variables are now domain variables. Let op denote an operator in the set $\{<, >, =, \le, \ge, \ne\}$ and let X and Y be domain variables. An atomic formula in DRC is one of the following:

- $(x_1, x_2, \ldots, x_n) \in$ Rel, where Rel is a relation with n attributes; each x_i, $1 \le i \le$ n is either a variable or a constant
- X op Y
- X op *constant*, or *constant* op X

A formula is recursively defined to be one of the following, where P and q are themselves formulas and p(X) denotes a formula in which the variable X appears:

- any atomic formula
- $\neg$ p, P $\wedge$ q, P V q, or p => q
- $\exists$X(p(X)), where X is a domain variable
- $\forall$ X(p(X)), where X is a domain variable

(Q1) Find the names of sailors who have reserved boat 103.

$$\{ \langle N \rangle \mid \exists I, T, A (\langle I, N, T, A \rangle \in Sailors$$

$$\wedge \exists\ Ir, Br, D(\langle Ir, Br, D \rangle \in Reserves \wedge Ir = I \wedge Br = 103))\}$$

(Q2) Find the names of sailors who have reserved a red boat.

$$\{\langle N\rangle \mid \exists I, T, A(\langle I, N, T, A\rangle \in Sailors$$

$$\wedge \exists \langle I, Br, D\rangle \in Reserves \wedge \exists \langle Br, BN, 'red'\rangle \in Boats)\}$$

(Q7) Find the names of sailors who have reserved at least two boats.

$$\{\langle N\rangle \mid \exists I, T, A(\langle I, N, T, A\rangle \in Sailors \wedge$$

$$\exists Br1, Br2, D1, D2 (\langle I, Br1, D1\rangle \in Reserves$$

$$\wedge \langle I, Br2, D2\rangle \in Reserves \wedge Br1 \neq Br2)$$

(Q9) Find the names of sailors who have reserved all boats.

$$\{\langle N\rangle \mid \exists I, T, A(\langle I, N, T, A\rangle \in Sailors \wedge$$

$$\forall B, BN, C(\neg(\langle B, BN, C\rangle \in Boats) \vee$$

$$(\exists \langle Ir, Br, D\rangle \in Reserves(I = Ir \wedge Br = B))))\}$$