Unit-IV

Introduction to the Relational Model:-

The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship. In relational model, the data and relationships are represented by collection of inter-related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records.

The relational model is today the primary data model for commercial data processing applications. It attained its primary position because of its simplicity, which eases the job of the programmer, compared to earlier data models such as the network model or the hierarchical model. In this, we first study the fundamentals of the relational model. A substantial theory exists for relational databases.

Integrity Constraint Over relations :-

Database integrity refers to the validity and consistency of stored data. Integrity is usually expressed in terms of constraints, which are consistency rules that the database is not permitted to violate. Constraints may apply to each attribute or they may apply to relationships between tables

Integrity constraints ensure that changes (update deletion, insertion) made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database.

Types of Integrity Constraints:- Various types of integrity constraints are-

a) Domain Integrity
b) Entity Integrity Constraint
c) Referential Integrity Constraint
d) Key Constraints

Domain Integrity- Domain integrity means the definition of a valid set of values for an attribute. You define data type, length or size, is null value allowed , is the value unique or not for an attribute ,the default value, the range (values in between) and/or specific values for the attribute.

Entity Integrity Constraint- This rule states that in any database relation value of attribute of a primary key can't be null.

Referential Integrity Constraint-It states that if a foreign key exists in a relation then either the foreign key value must match a primary key value of some tuple in its home relation or the foreign key value must be null.

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-

- Candidate key.
- Super key
- Primary key
- Foreign key

Enforcing Integrity constraints:- Data integrity refers to the correctness and completeness of data within a database. To enforce data integrity, you can constrain or restrict the data values that users can insert, delete, or update in the database. These mechanisms allow you to maintain these types of data integrity:

Requirement – requires that a table column must contain a valid value in every row; it cannot allow null values. The **create table** statement allows you to restrict null values for a column.

Check or validity – limits or restricts the data values inserted into a table column. You can use triggers or rules to enforce this type of integrity.

Uniqueness – no two table rows can have the same non-null values for one or more table columns. You can use indexes to enforce this integrity.

Referential – data inserted into a table column must already have matching data in another table column or another column in the same table. A single table can have up to 192 references.

Querying relational data:- A query is a request for data or information from a database table or combination of tables. This data may be generated as results returned by Structured Query Language (SQL) or as pictorials, graphs or complex results, e.g., trend analyses from data-mining tools.

One of several different query languages may be used to perform a range of simple to complex database queries. SQL, the most well-known and widely-used query language, is familiar to most database administrators (DBAs).

Logical data base Design

Database Schema

Database schema is the logical design of the database, and the **database instance** is a snapshot of the data in the database at a given instant in time. The concept of a relation corresponds to the programming-language notion of a variable, while the concept of a **relation schema** corresponds to the programming-language notion of type definition.

In general, a **relation schema** consists of a list of attributes and their corresponding domains. The concept of a **relation instance** corresponds to the programming-language notion of a value of a variable. The value of a given variable may change with time.

Table 4.5 Department Relation

Dept_name	Campus	No. of Rooms
Comp. Sci.	Science	7
Maths	Science	5
Tamil	Arts	4
Physics	Science	9
English	Arts	5
Bio-technology	Science	6
Physical Education	Education	8
Social Works	Arts	4

Similarly, the contents of a relation instance may change with time as the relation is updated. In contrast, the schema of a relation does not generally change. Although it is important to know the difference between a relation schema and a relation instance, we often use the same name, such

as *Faculty*, to refer to both the schema and the instance. Where required, we explicitly refer to the schema or to the instance, for example "the *Faculty* schema," or "an instance of the *Faculty* relation." However, where it is clear whether we mean the schema or the instance, we simply use the relation name.

Consider the *department* relation of Table 4.5. The schema for that relation is ***department* (*dept, campus, No. of Rooms*)**

Note that the attribute *dept* appears in both the *Faculty* schema and the *department* schema. This duplication is not a coincidence. Rather, using common attributes in relation schemas is one way of relating tuples of distinct relations.

For example, suppose we wish to find the information about all the Faculty members who work in the Science Campus. We look first at the *department* relation to find the *dept name* of all the departments housed in Science campus. Then, for each such department, we look in the *Faculty* relation to find the information about the Faculty associated with the corresponding *dept name*.

Let us continue with our university database example. Each course in a university may be offered multiple times, across different semesters, or even within a semester. We need a relation to describe each individual offering, or section, of the class. The schema is

***section* (*course id, semester, year, dept*)**

Table 4.6 Section Relation

Course_id	Semster	Year	Dept
CS-101	1	I	Comp. Sci.
MA-101	2	I	Maths
TA-102	1	I	Tamil
TA-204	3	II	Tamil
EN-101	1	I	English
MA-205	4	II	Maths
CS-204	4	II	Comp. Sci.

Table 4.6 shows a sample instance of the *section* relation. We need a relation to describe the association between Faculty and the class sections that they teach. The relation schema to describe this association is

***teaches* (*fac_id, course id, semester, dept*)**

Table 4.7 Teaches Relation

Fac_id	Course_id	Semster	Dept
F101	CS-101	1	Comp. Sci.
F306	MA-101	2	Maths
F204	TA-102	1	Tamil
F206	TA-204	3	Tamil
F401	EN-101	1	English
F305	MA-205	4	Maths
F103	CS-204	4	Comp. Sci.

Table 4.7 shows a sample instance of the *teaches* relation. As you can imagine, there are many more relations maintained in a real university database. In addition to those relations we have listed already, *Faculty*, *department*, *course*, *section*, *prereq*, and *teaches*, we use the following relations in this text:

- *student* (*ID, name, dept name, tot cred*)
- *advisor* (*s id, i id*)
- *takes* (*ID, course id, sec id, semester, year, grade*)

- *classroom* (*building*, *room number*, *capacity*)
- *time slot* (*time slot id*, *day*, *start time*, *end time*)

Schema Diagrams

A database schema, along with primary key and foreign key dependencies, can be depicted by **schema diagrams**. Figure 4.1 shows the schema diagram for our university organization. Each relation appears as a box, with the relation name at the top in blue, and the attributes listed inside the box. Primary key attributes are shown underlined. Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation

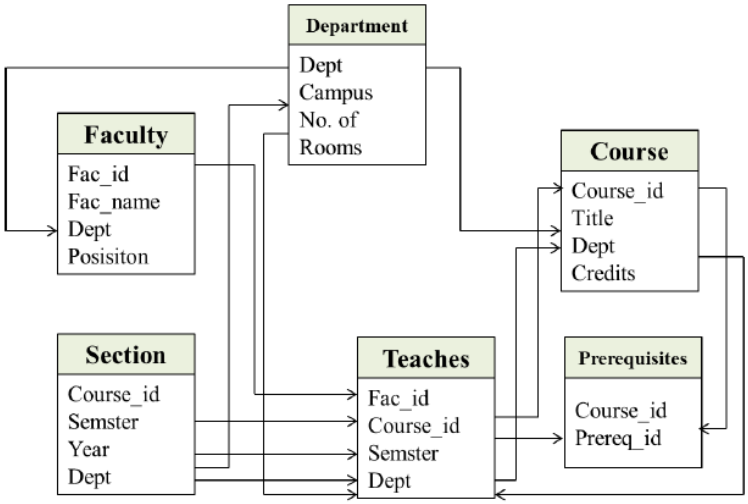


Figure 4.1 Schema Diagram for University Database

Referential integrity constraints other than foreign key constraints are not shown explicitly in schema diagrams. We will study a different diagrammatic representation called the entity-relationship diagram.

Introduction to views:-

- A view is nothing more than a SQL statement that is stored in the database with an associated name. A view is actually a composition of a table in the form of a predefined SQL query.
- A view can contain all rows of a table or select rows from a table. A view can be created from one or many tables which depends on the written SQL query to create a view.
- Views, which are a type of virtual tables allow users to do the following –
  - Structure data in a way that users or classes of users find natural or intuitive.
  - Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
  - Summarize data from various tables which can be used to generate reports.

Creating Views :- Database views are created using the **CREATE VIEW** statement. Views can be created from a single table, multiple tables or another view. To create a view, a user must have the appropriate system privilege according to the specific implementation.

The basic **CREATE VIEW** syntax is as follows −
CREATE VIEW view_name AS SELECT column1, column2..... FROM table_name WHERE [condition];

You can include multiple tables in your SELECT statement in a similar way as you use them in a normal SQL SELECT query.

Example :-
Consider the CUSTOMERS table having the following schema
Customers (cust_id, cust_name, age, address, mobile)

Following is an example to create a view from the CUSTOMERS table. This view would be used to have customer name and age from the CUSTOMERS table.

SQL > CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age FROM CUSTOMERS;

Now, you can query CUSTOMERS_VIEW in a similar way as you query an actual table. Following is an example for the same.
SQL > SELECT * FROM CUSTOMERS_VIEW;

Views with Check Option

The WITH CHECK OPTION is a CREATE VIEW statement option. The purpose of the WITH CHECK OPTION is to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view definition. If they do not satisfy the condition(s), the UPDATE or INSERT returns an error. The following code block has an example of creating same view CUSTOMERS_VIEW with the WITH CHECK OPTION.

CREATE VIEW CUSTOMERS_VIEW AS SELECT name, age FROM CUSTOMERS WHERE age IS NOT NULL WITH CHECK OPTION;

The WITH CHECK OPTION in this case should deny the entry of any NULL values in the view's AGE column, because the view is defined by data that does not have a NULL value in the AGE column.

Updating a View

A view can be updated under certain conditions which are given below −

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.

- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So, if a view satisfies all the above-mentioned rules then you can update that view. The following code block has an example to update the age of Ramesh.

SQL > UPDATE CUSTOMERS_VIEW SET AGE = 35 WHERE name = 'Ramesh';

This would ultimately update the base table CUSTOMERS and the same would reflect in the view itself. Now, try to query the base table and the SELECT statement would produce the following result.

Destroying /altering Tables and Views

Altering the Tables

The SQL ALTER TABLE command is used to add, delete or modify columns in an existing table. You should also use the ALTER TABLE command to add and drop various constraints on an existing table.

Syntax of an ALTER TABLE command to add a New Column in an existing table
ALTER TABLE table_name ADD column_name datatype;

Syntax of an ALTER TABLE command to DROP COLUMN in an existing table
ALTER TABLE table_name DROP COLUMN column_name;

Syntax of an ALTER TABLE command to change the DATA TYPE of a column in a table
ALTER TABLE table_name MODIFY COLUMN column_name datatype;

Syntax of an ALTER TABLE command to add a NOT NULL constraint to a column in a table
ALTER TABLE table_name MODIFY column_name datatype NOT NULL;

Syntax of ALTER TABLE to ADD UNIQUE CONSTRAINT to a table
ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint UNIQUE(column1, column2...);

Syntax of an ALTER TABLE command to ADD CHECK CONSTRAINT to a table
ALTER TABLE table_name ADD CONSTRAINT MyUniqueConstraint CHECK (CONDITION);

Syntax of an ALTER TABLE command to ADD PRIMARY KEY constraint to a table
ALTER TABLE table_name ADD CONSTRAINT MyPrimaryKey PRIMARY KEY (column1, column2...);

Syntax of an ALTER TABLE command to DROP CONSTRAINT from a table
ALTER TABLE table_name DROP CONSTRAINT MyUniqueConstraint;

Deleting Rows from a Table / View

Rows of data can be deleted from a table / view.
Following is an example to delete all records in both tables / views.

```
SQL > DELETE FROM CUSTOMERS;

SQL > DELETE FROM CUSTOMERS_VIEW;
```

The WHERE clause can be used to delete a record from table / view with conditions. Following is an example to delete a record having AGE = 22 in both tables / views.

```
SQL > DELETE FROM CUSTOMERS WHERE age = 22;
SQL > DELETE FROM CUSTOMERS_VIEW WHERE age = 22;
```

This would ultimately delete a row from the base table CUSTOMERS and the same would reflect in the view itself.

Dropping Tables / Views

The syntax to drop a table / view are given below −

```
DROP TABLE table_name;
DROP VIEW view_name;
```

Following is an example to drop the CUSTOMERS_VIEW and CUSTOMERS table.

```
DROP TABLE CUSTOMERS;
DROP VIEW CUSTOMERS_VIEW;
```

Relational Algebra and Relational Calculus

Relational Algebra:-

What is Relational Algebra (RA) ?

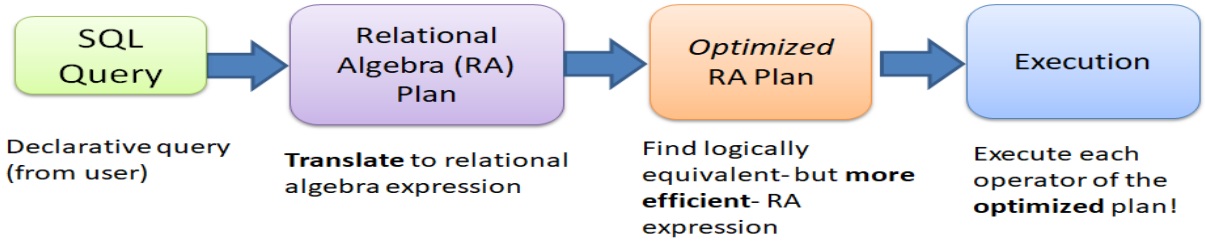
- An algebra whose operands are database tables or relations or variables that represent relations.
- Operators are designed to do the most common things that we need to do with relations in a database.
- The result is an algebra that can be used as a query language for relations

Why you should Learn "Relational Algebra" ?

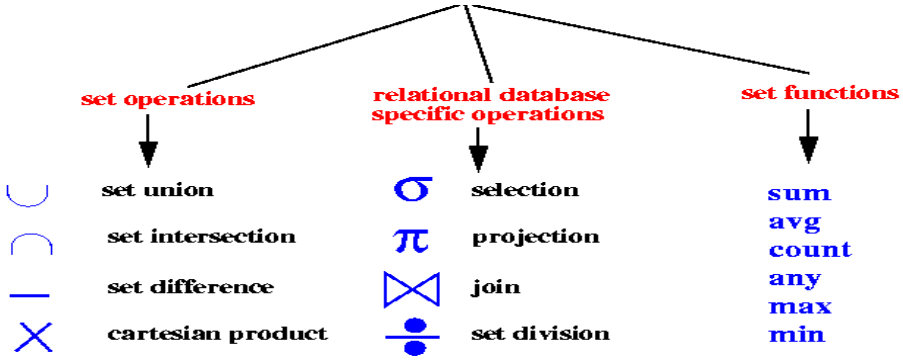
Relational Algebra -

- ✓ Is the Core of Relational Query Language, Example: **SQL**
- ✓ It Provides framework for Query implementation and optimization
- ✓ Is a ***mathematical language*** for manipulating relations.

How does a SQL engine work ?



Relational Algebra Operations:- The Relational Algebra operators are classified as follows:



- Relational Algebra is a procedural language consisting of a set of operations that take one or two relations as input and produce a new relation as their result.

Six basic operators

- select: σ
- project: Π
- Cartesian product: x
- rename: ρ
- union: ∪
- intersection: ∩
- difference: –
- division : ÷
- Join: ⋈

Select Operation (σ):-

- The **selec**t operation selects tuples that satisfy a given predicate (condition).
- $\sigma_{selection\ condition}(R)$
- Example: select those tuples of the *instructor* relation where the instructor is in the "Physics" department.

Query

$\sigma_{dept\_name="Physics"}(instructor)$

Result

ID	name	dept_name	salary
22222	Einstein	Physics	95000
33456	Gold	Physics	87000

- We allow comparisons using
  $=, \neq, >, \geq, <, \leq$
  in the selection predicate.
- We can combine several predicates into a larger predicate by using the connectives:
  $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
- Example: Find the instructors in Physics with a salary greater $90,000, we write:

  $\sigma_{dept\_name="Physics" \wedge salary>90,000}(instructor)$

- Then select predicate may include comparisons between two attributes.
  - Example, find all departments whose name is the same as their building name:
  - $\sigma_{dept\_name=building}(department)$

Project Operation - π (phi):-

It projects column(s) that satisfy a given predicate (condition) written as

Notation : $\pi_{A1,\ldots,An}(R)$

Where A_1, A_2 , A_n are attribute names of relation **R**.

- The result is defined as the relation of *k* columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets

Eg: $\pi_{name,dep\_num}(Student)$

Composition of Relational Operations:-

- The result of a relational-algebra operation is relation and therefore of relational-algebra operations can be composed together into a **relational-algebra expression**.
- Consider the query -- Find the names of all instructors in the Physics department.

$$\Pi_{name}(\sigma_{dept\_name = \text{“}Physics\text{”}}(instructor))$$

- Instead of giving the name of a relation as the argument of the projection operation, we give an expression that evaluates to a relation.

Cartesian-Product / cross Product / cross join Operation(X):-

Cartesian-Product / cross Product / cross join Operation

- The Cartesian-product operation (denoted by X) allows us to combine information from any two relations.
- Example: the Cartesian product of the relations *instructor* and t*eaches* is written as:

  instructor X *teaches*

- We construct a tuple of the result out of each possible pair of tuples: one from the *instructor* relation and one from the *teaches* relation (see next slide)
- Since the instructor *ID* appears in both relations we distinguish between these attribute by attaching to the attribute the name of the relation from which the attribute originally came.
  - *instructor.ID*
  - *teaches.ID*

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 2.1 The *instructor* relation.

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

The *teaches* relation.

The *instructor* x *teaches* table

Instructor.ID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	12121	FIN-201	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	15151	MU-199	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
12121	Wu	Finance	90000	10101	CS-101	1	Fall	2017
12121	Wu	Finance	90000	10101	CS-315	1	Spring	2018
12121	Wu	Finance	90000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
12121	Wu	Finance	90000	15151	MU-199	1	Spring	2018
12121	Wu	Finance	90000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
15151	Mozart	Music	40000	10101	CS-101	1	Fall	2017
15151	Mozart	Music	40000	10101	CS-315	1	Spring	2018
15151	Mozart	Music	40000	10101	CS-347	1	Fall	2017
15151	Mozart	Music	40000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
15151	Mozart	Music	40000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
22222	Einstein	Physics	95000	10101	CS-101	1	Fall	2017
22222	Einstein	Physics	95000	10101	CS-315	1	Spring	2018
22222	Einstein	Physics	95000	10101	CS-347	1	Fall	2017
22222	Einstein	Physics	95000	12121	FIN-201	1	Spring	2018
22222	Einstein	Physics	95000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...

- The Cartesian-Product

  instructor X *teaches*

  associates every tuple of instructor with every tuple of teaches.
  - Most of the resulting rows have information about instructors who did NOT teach a particular course.
- To get only those tuples of "*instructor* X *teaches* " that pertain to instructors and the courses that they taught, we write:

  $\sigma_{instructor.id = teaches.id}$ (*instructor* x *teaches*))
  - We get only those tuples of "*instructor* X *teaches*" that pertain to instructors and the courses that they taught.
- The result of this expression, shown in the next slide

- The table corresponding to:

$$\sigma_{instructor.id = teaches.id}(instructor \times teaches))$$

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

Figure 2.1 The *instructor* relation.

ID	course_id	sec_id	semester	year
10101	CS-101	1	Fall	2017
10101	CS-315	1	Spring	2018
10101	CS-347	1	Fall	2017
12121	FIN-201	1	Spring	2018
15151	MU-199	1	Spring	2018
22222	PHY-101	1	Fall	2017
32343	HIS-351	1	Spring	2018
45565	CS-101	1	Spring	2018
45565	CS-319	1	Spring	2018
76766	BIO-101	1	Summer	2017
76766	BIO-301	1	Summer	2018
83821	CS-190	1	Spring	2017
83821	CS-190	2	Spring	2017
83821	CS-319	2	Spring	2018
98345	EE-181	1	Spring	2017

The *teaches* relation.

InstructorID	name	dept_name	salary	teaches.ID	course_id	sec_id	semester	year
10101	Srinivasan	Comp. Sci.	65000	10101	CS-101	1	Fall	2017
10101	Srinivasan	Comp. Sci.	65000	10101	CS-315	1	Spring	2018
10101	Srinivasan	Comp. Sci.	65000	10101	CS-347	1	Fall	2017
12121	Wu	Finance	90000	12121	FIN-201	1	Spring	2018
15151	Mozart	Music	40000	15151	MU-199	1	Spring	2018
22222	Einstein	Physics	95000	22222	PHY-101	1	Fall	2017
32343	El Said	History	60000	32343	HIS-351	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-101	1	Spring	2018
45565	Katz	Comp. Sci.	75000	45565	CS-319	1	Spring	2018
76766	Crick	Biology	72000	76766	BIO-101	1	Summer	2017
76766	Crick	Biology	72000	76766	BIO-301	1	Summer	2018
83821	Brandt	Comp. Sci.	92000	83821	CS-190	1	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-190	2	Spring	2017
83821	Brandt	Comp. Sci.	92000	83821	CS-319	2	Spring	2018
98345	Kim	Elec. Eng.	80000	98345	EE-181	1	Spring	2017

Rename Operation - ρ (rho):-

- The results of relational-algebra expressions do not have a name that we can use to refer to them.
- The rename operator, ρ *(rho)* is provided for this purpose
- The expression:

$$\rho_x(E)$$

returns the result of expression *E* under the name *x*

- Another form of the rename operation:

$$\rho_{S(B1, B2, \ldots Bn)}(R) \quad \text{or} \quad \rho_S(R) \quad \text{or} \quad \rho_{(B1, B2, \ldots Bn)}(R)$$

Where the ρ (rho) is used to denote the RENAME operator, S is the new relation name, and B1, B2, … Bn are the new attribute names. The first expression renames both the relation and its attributes, the second renames the relation only, and the third renames the attributes only.

1) Query to rename the relation Student as Male Student and the attributes of Student – RollNo, SName as (Sno, Name).

$$\rho_{\text{MaleStudent(Sno, Name)}}\ \pi_{\text{RollNo, SName}}(\sigma_{\text{gender='Male'}}(\textbf{Student}))$$

2) Query to rename the attributes Name, Age of table Department to A,B.

$$\rho_{(A, B)}\ (\textbf{Department})$$

3) Query to rename the table name Project to Pro and its attributes to P, Q, R.

$$\rho_{\text{Pro(P, Q, R)}}\ (\textbf{Project})$$

4) Query to rename the first attribute of the table Student with attributes A, B, C to P.

$$\rho_{(P, B, C)}\ (\textbf{Student})$$

Assignment Operation (←):-

- It is convenient at times to write a relational algebra expression by assigning parts of it to temporary relation variables.
- The assignment operation is denoted by ← and works like assignment in a programming language.
- Example: Find all instructor in the "Physics" and "Music" department.

$$Physics \leftarrow \sigma_{dept\_name="Physics"}(instructor)$$
$$Music \leftarrow \sigma_{dept\_name="Music"}(instructor)$$

- With the assignment operation, a query can be written as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

$TEMP \leftarrow \sigma_{<Dno=5>}(EMPLOYEE)$

TEMP

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

Union Operation :-

- The union operation allows us to combine two relations
- Notation: $r \cup s$
- For $r \cup s$ to be valid.
  1. r, s must have the *same* **arity** (same number of attributes)
  2. The attribute domains must be **compatible** (example: 2nd column of r deals with the same type of values as does the 2nd column of s)
- Example: to find all courses taught in the Fall 2017 semester, or in the Spring 2018 semester, or in both

$$\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2017}(section)) \cup \Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2018}(section))$$

Result of:

$$\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2017}(section)) \cup \Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2018}(section))$$

course_id
CS-101
CS-315
CS-319
CS-347
FIN-201
HIS-351
MU-199
PHY-101

Set-Intersection Operation

- The set-intersection operation allows us to find tuples that are in both the input relations.
- Notation: $r \cap s$
- Assume:
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Example: Find the set of all courses taught in both the Fall 2017 and the Spring 2018 semesters.

$$\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2017}(section)) \cap \Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2018}(section))$$

  - Result

course_id
CS-101

Set Difference Operation

- The set-difference operation allows us to find tuples that are in one relation but are not in another.
- Notation $r - s$
- Set differences must be taken between **compatible** relations.
  - r and s must have the same arity
  - attribute domains of r and s must be compatible
- Example: to find all courses taught in the Fall 2017 semester, but not in the Spring 2018 semester

$$\Pi_{course\_id}(\sigma_{semester="Fall" \wedge year=2017}(section)) - \Pi_{course\_id}(\sigma_{semester="Spring" \wedge year=2018}(section))$$

course_id
CS-347
PHY-101

Division Operation (÷)

- The Division is not often used in relational queries.
- It may be used to solve certain complicated problems occasionally.
- The division operator is used for queries which involve the 'all'.
- It is represented with the symbol ÷
- Notation : R1 ÷ R2
- The above notation represents tuples of R1 associated with all tuples of R2

Consider two relations R and S. Assume that R has only two attributes x and y and S has just one attribute y with the same domain as in R.

This is to ensure that the degree of the numerator is more than the degree of the denominator.

R1

Name	Course
System	Btech
Database	Mtech
Database	Btech
Algebra	Btech
Algebra	B.Sc

R2

Course
Btech
Mtech

R1 ÷ R2

Name
database

Employee

Name	Eno	Pno
John	123	P1
Smith	123	P2
A	121	P3

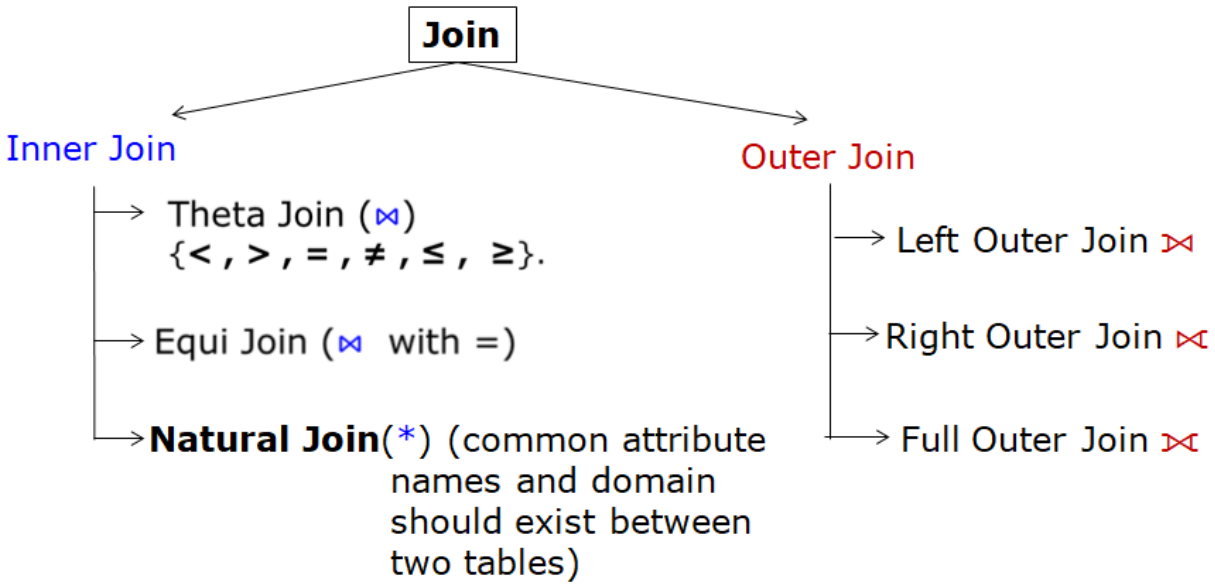
Works

Eno	Pno	Pname
123	P1	Market
123	P2	Sales

Employee ÷ Works

Eno
123

Join Operations



Inner Joins:- This is the most common type of join. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. There are 3 types of Inner joins :

- ✓ Theta Join
- ✓ Equi join
- ✓ Natural Join

Theta Join (θ)

- Theta join combines tuples from different relations provided they satisfy the theta condition.
- The join condition is denoted by the symbol **θ**.
- Notation: R1 $\bowtie_{\theta}$ R2
- The join condition can be any of the comparison operators {=, <, <=, >, >=, ≠}

Student		
SID	Name	Std
101	Alex	10
102	Maria	11
103	John	12

Subjects	
Class	Subject
10	Math
10	English
11	Music
11	Sports

Student_Detail:

STUDENT $\bowtie_{\text{Student.Std} = \text{Subjects.Class}}$ SUBJECTS

Student_detail				
SID	Name	Std	Class	Subject
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Car

CarModel	CarPrice
CarA	20,000
CarB	30,000
CarC	50,000

Boat

BoatModel	BoatPrice
Boat1	10,000
Boat2	40,000
Boat3	60,000

$Car \bowtie_{CarPrice \geq BoatPrice} Boat$

CarModel	CarPrice	BoatModel	BoatPrice
CarA	20,000	Boat1	10,000
CarB	30,000	Boat1	10,000
CarC	50,000	Boat1	10,000
CarC	50,000	Boat2	40,000

Equi Join

It is the most common join. It is based on matched data as per the equality condition. The equi join uses the comparison operator(=).

table1

num	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

num	M	ID	N
2	b	2	p

Select *
From table1, table 2
Where num=id;

Relational algebra expression

$$result \leftarrow table1 \bowtie_{num=id} table2$$

num	M	ID	N
2	b	2	p

Natural Join

A natural join is the set of tuples of all combinations in R and S that are equal on their common attribute names. It is denoted by ⋈.
Notation: R1 ⋈ R2

result <- table1 ⋈ table2 OR result <- table1 * table2

table1

ID	M
1	a
2	b
4	c

table2

ID	N
2	p
3	q
5	r

result

ID	M	N
2	b	p

- ✓ Natural join does not use any comparison operator.
- ✓ We can perform a Natural Join only if there exists atleast one common attribute between two relations.
- ✓ In addition, the attributes must have the same name and domain.
- ✓ Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Outer Joins

- An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation. Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation.
- It is an extension of natural join to deal with missing values of relation
- These return matched values and unmatched values from either or both tables
- These are of 3 types
  - Left Outer Join ⟕
  - Right Outer Join ⟖
  - Full Outer Join ⟗

Left Outer Join ⟕

- ✓ Left Outer Join is similar to a natural join, but it returns all the rows of the table on the left side of the join and matching rows for the table on the right side of join.
- ✓ The rows for which there is no matching row on right side, the result-set will contain null. Left Outer Join is also known as Left Join.
- ✓ Let R and S be two relations and the Left Outer Join (⟕) can be written as $R \text{ ⟕ } S$

Table R1

RegNo	Branch	Section
1	CSE	A
2	ECE	B
3	CIVIL	A
4	IT	B
5	IT	A

Table R2

Name	Regno
Bhanu	2
Priya	4
Hari	7

R1 ⟕ R2

RegNo	Branch	Section	Name	Regno
2	ECE	B	Bhanu	2
4	IT	B	Priya	4
1	-	-	NULL	NULL
3	-	-	NULL	NULL
5	-	-	NULL	NULL

Right Outer Join ⟖

- Right Outer Join is similar to a natural join, but it returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join.
- The rows for which there is no matching row on left side, the result-set will contain null.
- Right Outer Join is also known as Right Join.
- Let R and S be two relations and the Right Outer Join (⟖) can be written as : $R ⟖ S$

Table R1

RegNo	Branch	Section
1	CSE	A
2	ECE	B
3	CIVIL	A
4	IT	B
5	IT	A

Table R2

Name	Regno
Bhanu	2
Priya	4
Hari	7

R1 ⟖ R2

RegNo	Branch	Section	Name	Regno
2	ECE	B	Bhanu	2
4	IT	B	Priya	4
NULL	NULL	NULL	Hari	7

Full Outer Join ⟗

- ✓ Full Outer Join is similar to a natural join. It creates the result-set by combining result of both Left Outer Join and Right Outer Join.
- ✓ The result-set will contain all the rows from both the tables.
- ✓ The rows for which there is no matching, the result-set will contain NULL values. Full Outer Join is also known as Full Join.
- ✓ Let R and S be two relations and the Right Outer Join (⟗) can be written as : $R ⟗ S$

Table R1

RegNo	Branch	Section
1	CSE	A
2	ECE	B
3	CIVIL	A
4	IT	B
5	IT	A

Table R2

Name	Regno
Bhanu	2
Priya	4
Hari	7

R1 ⟗ R2

RegNo	Branch	Section	Name	Regno
2	ECE	B	Bhanu	2
4	IT	B	Priya	4
1	-	-	NULL	NULL
3	-	-	NULL	NULL
5	-	-	NULL	NULL
NULL	NULL	NULL	Hari	7

Relational Calculus:-

- Relational calculus is a *non-procedural query language*.
- It uses mathematical predicate calculus instead of algebra.
- Relational calculus provides the description about the query to get the result whereas relational algebra gives the method to get the result.
- Relational calculus informs the system *what* to do with the relation whereas relational algebra informs the system *how* to do with the relation.
- This differs from relational algebra, which is procedural query language, where we must write a sequence of operations to specify a retrieval request.
- It has been shown that any retrieval that can be specified in the basic relational algebra can also be specified in relational calculus, and vice versa; in other words, the expressive power of the two languages is identical.
- There are two types of relational calculus
  - ✓ Tuple Relational Calculus (TRC)
  - ✓ Domain Relational Calculus (DRC).

Tuple Relational Calculus (TRC):-

Tuple Relational Calculus (TRC)

- A Tuple Relational Calculus is a non procedural query language which specifies to select a number of **tuple variables** in a relation.
- It can select the tuples with **range** of values or tuples for certain attribute values etc.
- The language SQL is based on tuple calculus
- The resulting relation can have one or more tuples. It is denoted as below:
  { t | condition (t) }
- This is also known as expression of relational calculus
- Where *t* is the resulting tuple (tuple variable), *condition (t)* is the conditional expression used to fetch *t*. The result of such query is the set of all tuples *t* that satisfy *condition (t)*.

For example, to find all employees whose salary is above Rs 50000, we can write the following tuple calculus expression:
{t | EMPLOYEE (t) AND t.Salary>50000}

The above statement implies that it selects all the tuples from EMPLOYEE relation such that resulting employee tuples will have salary greater than 50000. It is example of selecting a range of values.

{t | EMPLOYEE (t) AND t.DEPT_ID = 10}
The above statement selects all the tuples of employee name who work for Department 10.
To retrieve only some of the attributes like the first name and last name and whose salary > 50000 , we write as follows:

{t.Fname, t.Lname | EMPLOYEE (t) AND t.Salary>50000}

The Existential and Universal Quantifiers

✓Two special symbols called quantifiers can appear in condition; these are the universal quantifier (∀) and the existential quantifier (∃).
✓∀ is called the universal or "for all" quantifier because every tuple in "the universe of" tuples must make F true to make the quantified formula true.
✓∃ is called the existential or "there exists" quantifier because any tuple that exists in "the universe of" tuples may make F true to make the quantified formula true.

✓A tuple variable t is bound if it is quantified, meaning that it appears in an (∀ t) or (∃ t) clause; otherwise, it is free.

✓If F is a formula/condition, then so are (∃ t)(F) and (∀ t)(F), where t is a tuple variable.
- ➢The formula (∃ t)(F) is true if the formula F evaluates to true for some (at least one) tuple assigned to free occurrences of t in F; otherwise (∃ t)(F) is false.
- ➢The formula (∀ t)(F) is true if the formula F evaluates to true for every tuple (in the universe) assigned to free occurrences of t in F; otherwise (∀ t)(F) is false.

➢Retrieve the name and address of all employees who work for the 'Research' department. The query can be expressed as :

{t.FNAME, t.LNAME, t.ADDRESS | EMPLOYEE(t) and (∃ d) (DEPARTMENT(d) and d.DNAME='Research' and d.DNUMBER=t.DNO) }

➢The only *free tuple variables* in a relational calculus expression should be those that appear to the left of the bar (|).

➢In above query, t is the only free variable; it is then *bound successively* to each tuple.

➢If a tuple *satisfies the conditions* specified in the query, the attributes FNAME, LNAME, and ADDRESS are retrieved for each such tuple.

➢The conditions EMPLOYEE (t) and DEPARTMENT(d) specify the range relations for t and d..

➢The condition d.DNAME = 'Research' is a selection condition and corresponds to a SELECT operation in the relational algebra, whereas the condition d.DNUMBER = t.DNO is a JOIN condition.

Domain Relational Calculus (DRC)

- ✓ In domain relational calculus, filtering variable uses the domain of attributes.
- ✓ Domain relational calculus uses the same operators as tuple calculus.
- ✓ It uses logical connectives ∧ (and), ∨ (or) and ¬ (not).
- ✓ It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.
- ✓ The QBE or Query by example is a query language related to domain relational calculus.

✓The language called QBE (Query-By-Example) that is related to domain calculus was developed almost concurrently to SQL at IBM Research, Yorktown Heights, New York.

✓Domain calculus was thought of as a way to explain what QBE does.

✓Domain calculus differs from tuple calculus in the type of variables used in formulas.

✓Rather than having variables range over tuples, the variables range over single values from domains of attributes.

✓To form a relation of degree n for a query result, we must have n of these domain variables— one for each attribute.

✓ **Notation:** { a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

where a1, a2 are attributes and P stands for formula built by inner attributes

Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

Query :

{uv | (∃ q) (∃ r) (∃ s) (∃ t) (∃ w) (∃ x) (∃ y) (∃ z)
(EMPLOYEE(qrstuvwxyz) and q='John' and r='B' and s='Smith')}

Abbreviated notation EMPLOYEE(qrstuvwxyz) uses the variables without the separating commas: **EMPLOYEE(q,r,s,t,u,v,w,x,y,z)**

✓Ten variables for the employee relation are needed, one to range over the domain of each attribute in order.

✓Of the ten variables q, r, s, . . ., z, only u and v are free.

✓Specify the requested attributes, BDATE and ADDRESS, by the free domain variables u for BDATE and v for ADDRESS.

✓Specify the condition for selecting a tuple following the bar (|)— namely, that the sequence of values assigned to the variables qrstuvwxyz be a tuple of the employee relation and that the values for q (FNAME), r (MINIT), and s (LNAME) be 'John', 'B', and 'Smith', respectively.

****