**Conceptual Design**

* E-R model: entities, attributes and its types, Relationship, Relationship sets, Generalization, Specialization, Aggregation
* Relational Model:
* Relation, Domain, Tuples,
* Types of keys,
* Relational Integrity Rules,
* Relational Algebra operations-
* Select, Project, Cartesian Product, Union, Set difference
* Join & types, Natural Join, Outer Join, lossless joins.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* **Relational database design:**

A relational database stores its data in 2-dimensional tables. They are used to establish and identify relation between tables.

* **Key:**

It is used for identifying unique rows from table.

1. To create relationships between two tables.
2. To maintain uniqueness in a table.
3. To keep consistent and valid data in database.
4. helps in fast data retrieval by enabling indexes on columns

* **Types of Keys**

**Simple Key:**A key which has the single attribute is known as a simple key

**Composite key:**A key which contain two or more attributes is called as Composite Key.

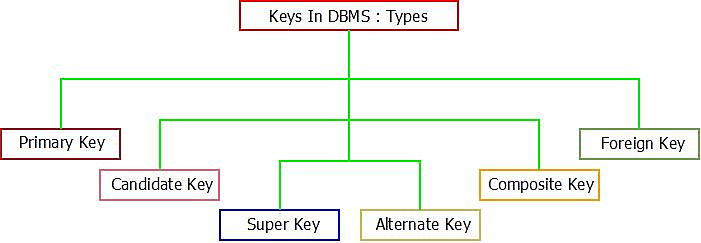
|  |  |  |  |
| --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **MOBILE\_NO** |
| 1 | RAM | DELHI | 9455123451 |
| 2 | RAMESH | GURGAON | 9652431543 |
| 3 | SUJIT | ROHTAK | 9156253131 |
| 4 | SURESH | DELHI | 9876543210 |

**Record**

**Field**

In the above table, each column is a field and each row is a record.

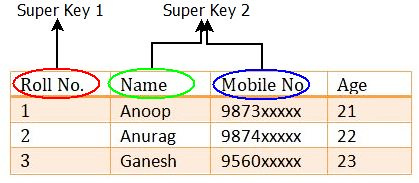
**Different types of keys are:**



### ****Super key****

A super key is a set of one or more attributes (columns) that are used to uniquely identify a row in a table. It includes only those fields that have unique values. Super key may hold some additional columns which are not strictly required to uniquely identify each row. Super Key is a superset of Candidate key and subset of Primary key, Unique key, Alternate key

**Example:** In the student table, roll\_no, name, address, phone, age are super keys. All of the following sets of super key are able to uniquely identify a row from the table.



Super key can be a single attribute (Roll No.)or a collection of multiple attributes (Name, Mobile

No) which can help in identification of record of every student.

### ****Candidate key****

### Candidate key is a set of one or more than one columns (attributes) which uniquely

### identifies each record in a table, candidate key should not have any redundant column values (repetition of cell values). Candidate key is a subset of super key. The candidate keys are as strong as the primary key. A column of candidate key select from a set of super key. “Every candidate key is a super key but, every super key may or may not be a candidate key.”

### Example: Student table has four attributes: roll\_no, name, mobile\_no, age. Here roll\_no & mobile\_no will be having unique values but Name and age can have duplicate values as more than one student can have same name.

Select the candidate keys from the following set of super keys-

* {roll\_no} – No redundant attributes
* {mobile\_no} – No redundant attributes
* {roll\_no, mobile\_no} – No redundant attributes
* {roll\_no, name} - Redundant attribute name
* {roll\_no, age} - Redundant attribute address
* {roll\_no, mobile\_no, age} - Redundant attribute age
* {roll\_no, mobile\_no, name}
* {mobile\_no, name}

|  |  |  |  |
| --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | AGE | **MOBILE\_NO** |
| 1 | RAM | 20 | 9455123451 |
| 2 | RAMESH | 22 | 9652431543 |
| 3 | SUJIT | 20 | 9156253131 |
| 4 | SURESH | 21 | 9876543210 |

**Candidate key**

Two set of super keys are chosen from the above sets as there are no redundant columns in these sets. The **candidate keys** we have selected are:

* {roll\_no}
* {mobile\_no}

Only these two sets are candidate keys as all other sets are having redundant attributes that are not necessary for unique identification. A [primary key](https://beginnersbook.com/2015/04/primary-key-in-dbms/) is selected from the set of candidate keys. We can either have roll\_no or mobile\_no as primary key.

**Candidate key it must pass certain criteria:**

* It must contain unique values
* It does not contain null values
* It contains the minimum number of fields to ensure uniqueness
* It must uniquely identify each record in the table

### ****Primary key****

A Primary key is selected from a set of candidate keys. A column or group of columns

in a table which helps us to uniquely identify every row in that table is called a primary key. Primary key are as follows:

1. Only one primary key can be assigned to a table
2. The value in a primary key column must be unique
3. Primary key cannot accept null and duplicate values
4. The value in a primary key column cannot be modified or updated if any foreign key refers to that primary key.

Primary key is reference for the table and used throughout the database to establish relationships with other tables. In below table the student Address and Name cannot act as primary key because this column values may be repeated.

|  |  |  |  |
| --- | --- | --- | --- |
| **ROLL\_NO** | **NAME** | **ADDRESS** | **PHONE** |
| 1 | RAM | DELHI | 9455123451 |
| 2 | RAMESH | GURGAON | 9652431543 |
| 3 | SUJIT | ROHTAK | 9156253131 |
| 4 | SURESH | DELHI | 9876543210 |

**Primary Key**

Only **Roll\_No** is unique in the above table, so it is selected as primary key. Mobile\_no can also be selected as a primary key.

### ****Composite key****

The primary key consists of two or more attributes is known as composite key. A key having combination of more than one attribute to uniquely identify rows in a table is called a composite key. It is also known as compound key. **Any part of the compound key can be a foreign key, but the composite key may or maybe not a part of the foreign key.**

|  |  |  |
| --- | --- | --- |
| **ROLL\_NO** | **SUBJECT** | **MARKS** |
| 1 | ENGLISH | 56 |
| 1 | MATHS | 78 |
| 2 | ENGLISH | 84 |
| 3 | COMPUTER | 94 |

**Example:**

**Roll No.** and **subject** is combined to uniquely identify the record in relation.

**Composite Key**

### Secondary key or Alternative Key

The key other than primary keys are called as secondary or alternative keys. The remaining keys which are not selected as primary key are called as alternate key. A table may have one or more choices for the primary key. Secondary key is required for the indexing purpose for better and faster searching. DBA can choose any of primary key.

|  |  |  |  |
| --- | --- | --- | --- |
| **REG\_NO** | **ROLL\_NO** | **NAME** | **CLASS** |
| 10 | 1 | RAM | MSc |
| 20 | 2 | RAMESH | MCA |
| 30 | 3 | SUJIT | BCA |

**Alternative Key**

**Primary Key**

Since we have selected Reg\_No as primary key, the remaining key Roll\_No would be

called alternative or secondary key.

### Foreign key

### Foreign keys are the column of a table that points to the [primary key](https://beginnersbook.com/2015/04/primary-key-in-dbms/) column of another table. They act as a cross-reference between tables. A foreign key is a column which is added to create a relationship with another table. The purpose of the foreign key is to ensure referential integrity of the data.

### The values of the FOREIGN KEY columns have to match with corresponding primary key columns values of a row in the referenced table.

### The RollNo attribute in Parent relation is used as primary key. The RollNo attribute in child relation is used as foreign key. It refers to RollNo attribute in Parent relation.

|  |  |  |
| --- | --- | --- |
| **ROLL\_NO** | **NAME** | **CLASS** |
| 1 | RAM | MSc |
| 2 | RAMESH | MCA |
| 3 | SUJIT | BCA |

**Foreign Key**  **Composite Primary Key**

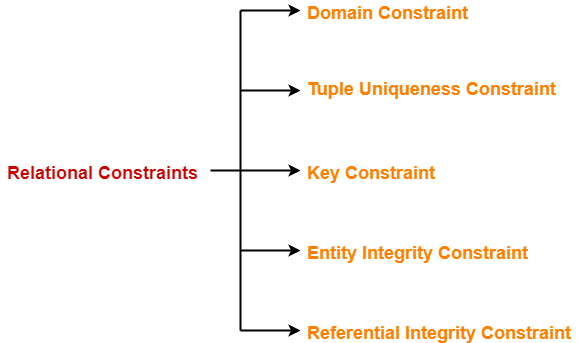
|  |  |  |  |
| --- | --- | --- | --- |
| **ROLL\_NO** | **SID** | **SUBJECT** | **MARKS** |
| 1 | 101 | MATHS | 78 |
| 2 | 102 | ENGLISH | 84 |
| 3 | 103 | COMPUTER | 94 |

**Primary key Reference use other table**

**Parent Relation Child Relation**

* **Relational Integrity Rules**

These are some rules or constraints applied on the database to keep data stable, accurate & consistent. We have to follow some integrity rules or integrity constraints.



**1) Entity Integrity Rule (Integrity Rule 1)**

Entity Integrity guarantee that there are no duplicate records within the table. Entity Integrity constraint uniquely identifies each row in table.

Entity Integrity must have two properties for primary keys:

* Primary key must be unique for each row in the table that having no same value.
* Primary key fields cannot be null.

**A duplicate value in primary key is invalid.**

**2) Referential Integrity Rule (Integrity Rule 2)**

A referential integrity rule is specified between two tables. It is linked data between two

or more [tables](https://database.guide/what-is-a-table/). **A foreign key (child table) value is valid reference, if it is existing in the**

**primary key table (parent table).** A foreign key can be either null.

* A column or collection of column in one table whose values must match the primary key in the other table is known as a foreign key.
* If a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2. **Cust\_id is primary key**

**3) Domain Constraints**

The restrictions applied on column (domain) for a valid set of values are known as domain constraints. Database keep consistency through applying restrictions on every value of column. These restrictions include data types and length (integer, varchar, char, time format, date format etc.), size of variable, checks (like value not null) etc.

**Example:** create table Customer (Cust\_id char (4), Cust\_Name char (20),

Cust\_Address char (20), Cust\_Age integer(3)Check (age>18),

Cust\_Mobile\_No integer, Primary key (Cust\_id),);



**4) Tuple Uniqueness Constraints**

Tuple Uniqueness constraint specifies that all the tuples must be necessarily unique in any relation. Duplicate tuples within a single relation are not allowed.

|  |  |  |
| --- | --- | --- |
| STU\_ID | Name | Age |
| S001 | Akshay | 20 |
| S002 | Abhishek | 21 |
| S003 | Shashank | 20 |
| S004 | Rahul | 20 |

This relation satisfies the tuple uniqueness constraint since here all the tuples are unique

* **Relational Algebra operations:**

****The relational algebra is a theoretical procedural query language associated with the relational model. It consists of a set of operations that take one or two relations (tables) as input and produce a new relation, on the request of the user to retrieve the specific information, as the output.

The relational algebra is very important due to many reasons.

* It provides a basic operation for relational model
* It is used as basis for implementing and optimizing queries in RDBMS's.
* The basic concepts of relational algebra are incorporated into the SQL language

The relational algebra uses various logical operator [Λ(and), V(or), ¬(not)] and comparison operators (<, <=, =, \*, >=, >) to construct composite and complex queries.

Query languages are of two types,

1. **Procedural language**: the user has to describe the specific procedure to retrieve the information from the database. Example: The Relational Algebra is a procedural language.
2. **Non-procedural language:** the user retrieves the information from the database without describing the specific procedure to retrieve it. Example: Tuple Relational Calculus and

Domain Relational Calculus are non-procedural languages.

* **Operations in Relational Algebra**

Relational Algebra Fundamental operations are divided into two groups

 **Basic set-oriented operations (Simple operations)**

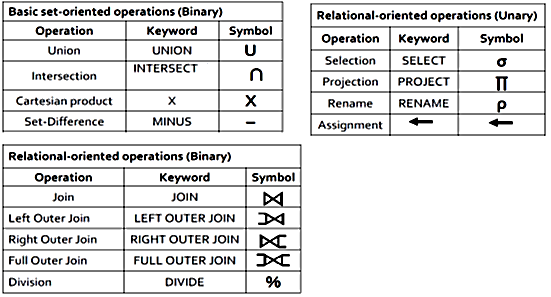
* Union (binary operations)
* Intersection (binary operations)
* Set difference (binary operations)
* Cartesian product (binary operations)

 **Relational-oriented operations (Special operations)**

* Selection (unary operations)
* Projection (unary operations)
* Joins (binary operations)
* Division (binary operations)
* Rename (binary operations)
* Assignment (binary operations)

The Selection, Projection and Rename operations are called **unary operations** because they operate only on one relation.

The other operations operate on pairs of relations are called **binary operations**



This database contains two Tables EMPLOYEE and STUDENT and the relationship is that employees can also a student and vice-versa.

**Employee Student**

|  |  |  |  |
| --- | --- | --- | --- |
| **EID** | **Name** | **Salary** | **Join year** |
| 1 | John | 16000 | 1994 |
| 2 | Ramesh | 5000 | 2000 |
| 3 | Smith | 8000 | 2001 |
| 4 | Jack | 6000 | 2000 |
| 5 | Nilesh | 15000 | 1998 |

|  |  |  |
| --- | --- | --- |
| **SID** | **Name** | **Fees** |
| 101 | Smith | 1000 |
| 102 | Vijay | 950 |
| 103 | Gaurav | 2000 |
| 104 | Nilesh | 1500 |
| 105 | John | 950 |

* **Basic set-oriented operations (Simple operations)**
* **Union operation (U):**

The union operation is a binary operation that is used to find union of tables. It combines

the similar columns from two tables into one resultant table. Duplicate values are removed.

Here relations are considered as sets. It is denoted by (U).A union

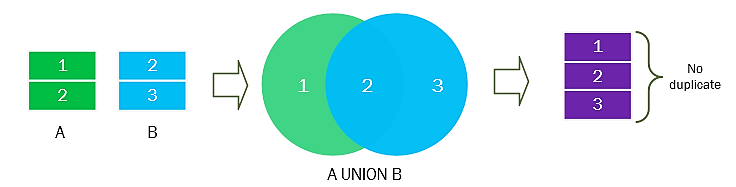
operation R U S to be valid, we require that two necessary conditions must be satisfied:

* Both relations have same number of attribute i.e. Degree of relations are same
* Data types of attributes in both the relations should be same.

Above two conditions, If R and S are two relations, which are union well-matched,

resulting tuples of relation either in R or S or both. There are no duplicate tuples. The union operator is: R **U** S = S **U** R

**Examples:**If you want to find the names of all employees and names of all students jointly then **∏**Name (Employee) **U ∏**Name (Student)

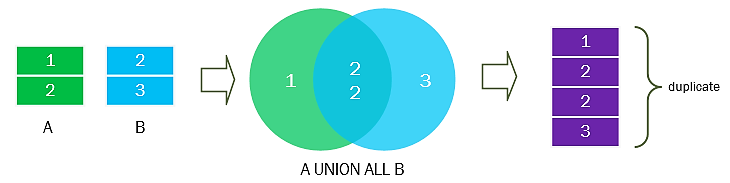
**** Select name from employee **UNION** Select name from student;

|  |
| --- |
| **Name** |
| John |
| Ramesh |
| Smith |
| Jack |
| Nilesh |
| Vijay |
| Gaurav |

* **UNION ALL**

This operation is also similar to UNION, but it does not eliminate the duplicate records. It shows all the records from both the tables. All other features are same as UNION.

**Example:** Select name from employee **UNION ALL** Select name from student;

****

* **Intersection operation (∩ )**

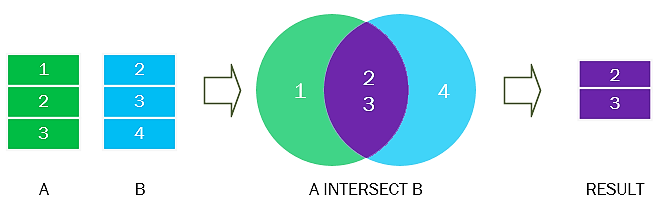
Intersection is used to find common tuples between two tables. It is denoted by (∩).Rules of Union operations are also applicable to intersection operation i.e. same degree and same domain. If R and S are two relations and we take intersection of these two relations then the resulting relation would be the set of tuples, which are in both R and S. R **∩** S = S **∩** R

**Examples:** find all the employees from Relation Employee those are also students.

|  |
| --- |
| **Name** |
| John |
| Smith |
| Nilesh |

**∏**Name (Employee) **∩ ∏**Name (Student)

Select name from employee **INTERSECT** Select name from student;

****

* **Set-difference operation (—):**

Set-difference operation is a binary operation which is used find tuples that are present in one (first) table but not into other (second) table. It is denoted by (—).It removes the common tuples of two relations and produces a new relation having rest of the tuples of first relation.

**Examples:** If you want the names of those employees that are not students, then the

**∏**Name (Employee) **— ∏**Name (Student)

Select name from employee **MINUS** Select name from student;

|  |
| --- |
| **Name** |
| Ramesh |
| Jack |

* **Cartesian product operation (X):**

Cartesian product is a binary operation which is used combine information of any two

relations. It is denoted by (X).Suppose a relation *R1* is having *m* tuples and other

relation *R2* is having *n* tuples then *R1 x R2* has *m x n* tuples. The Cartesian product needs not to be union compatible. It means they can be of different degree.



Select \* from Paper, Student;



* **Selection or Restriction operation (σ):**
* It is used to select certain rows or tuples of a table, so it performs on the table horizontally. It is denoted by sigma (**σ**).
* This command works on a single table and takes rows that meet a specified condition, copying them into a new table.
* As a result of this operation a new table is formed, without changing the original table i.e. Degree of table.
* Syntax: Selection condition (relation\_name)

**σ** <selection condition> (R)

* Selection condition is the Boolean expression on the attributes of relation R. The selection condition is represented as

**σ** <Attribute name> <comparison operator> <constant value> or

**σ** <Attribute name> <comparison operator> <attribute name> Where,

* < Attribute name > is the name of an attribute (column) of the relation R.
* < Comparison operator > is one of the comparison operators (=, #, <, <=,>,>=).
* < Constant value > is the constant value from the attribute domain.

Clauses are connected by Boolean operators AND, OR and NOT.

## Examples: Display all employees having salary greater than 9,000 from Employee.

## σ Salary> 9000 (Employee)

Select \* from employee where salary > 9000;

|  |  |  |  |
| --- | --- | --- | --- |
| **EID** | **Name** | **Salary** | **Joinyear** |
| 1 | John | 16000 | 1994 |
| 5 | Nilesh | 15000 | 1998 |

1. Display employees joined in year 2000.

**σ** Joinyear=2000 (Employee)

Select \* from employee where joinyear=2000;

|  |  |  |  |
| --- | --- | --- | --- |
| **EID** | **Name** | **Salary** | **Joinyear** |
| 2 | Ramesh | 5000 | 2000 |
| 4 | Jack | 6000 | 2000 |

1. Select the tuples for all employees whose joining year is 1998 or salary is greater than 10000. **σ** (Joinyear=1998) OR (Salary>10000) (Employee)

Select \* from employee where Joinyear=1998 OR Salary>10000;

|  |  |  |  |
| --- | --- | --- | --- |
| **EID** | **Name** | **Salary** | **Joinyear** |
| 5 | Nilesh | 15000 | 1998 |

Select the tuples for all employees whose joining year is 1994 and salary is greater than 10000. **σ** (Joinyear=1994) AND (Salary>10000) (Employee)

Select \* from employee where Joinyear = 1994 AND Salary > 10000;

|  |  |  |  |
| --- | --- | --- | --- |
| **EID** | **Name** | **Salary** | **Joinyear** |
| 1E | John | 16000 | 1994 |

1. Display employee have EID is 3E.

**σ** EID=’3E’ (Employee)

|  |  |  |  |
| --- | --- | --- | --- |
| **EID** | **Name** | **Salary** | **Joinyear** |
| 3E | Smith | 8000 | 2001 |

* **The Project Operation (∏)**
  + The project operation selects certain columns (Attribute) from a table while remove others. It is denoted by sigma (**∏**).
  + Create a new table by selecting only specified attributes of the existing table.
  + Remove duplicate tuples in the newly formed relation.
  + representation of Projection operation : **∏<Attribute list > (R)**

# Examples: Display all employee Names with Salary and Joinyear

**∏** Name, Salary, Joinyear (Employee)

Select Name, Salary, Joinyear from employee

|  |  |  |
| --- | --- | --- |
| **Name** | **Salary** | **Joinyear** |
| John | 16000 | 1994 |
| Ramesh | 5000 | 2000 |
| Smith | 8000 | 2001 |
| Jack | 6000 | 2000 |
| Nilesh | 15000 | 1998 |

## Composition of Select and Project Operations

The relational operations select and project can be combined to form a complex query.

**Example:** Display the Name of Employee having Salary greater than 8000.

**∏** Name (**σ** Salary > 8000 (Employee)) or **σ** Salary > 8000 (∏ Name (Employee))

|  |
| --- |
| **Name** |
| John |
| Smith |
| Nilesh |

Display the EID and Name of Employee having Joining year is 2000.

**∏** EID, Name (**σ** Joinyear=’2000’ (Employee))

|  |  |
| --- | --- |
| **EID** | **Name** |
| 2 | Ramesh |
| 4 | Jack |

* **The Rename Operation (ρ)**
* This is a unary operator which changes attribute names for a relation without changing any values.
* The rename operation provides database designers to rename the output relation. Notation: ρ OldName→ NewName(r)
* Example, ρ Name → Emp\_name(Empolyee)
* If there are two or more attributes involved in a renaming operation, then ordering is meaningful: e.g., ρ Eid,Salary → Emp\_id,Payment(Employees)

# Mechanisms for joining relations (Join Operation) -inner joins, outer joins and its types

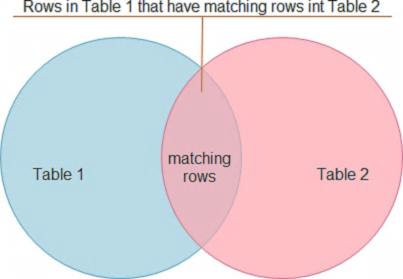
* The process of linking tables is called joining. SQL handles queries across more than one table through the use of JOINs.
* JOINs are clauses in SQL statements that link two tables together, usually based on the keys that define the relationship between those two tables.
* The purpose of Join statement is to combine data or rows from two or more tables based on a common field between them.
* SQL can get data from several related tables by performing either a physical or virtual join on the tables.
* The WHERE Clause is mostly used to perform the JOIN function with two or more tables have columns.

**Join Syntax:** SELECT [table\_name.column\_name1, table\_name.column\_name2,….] FROM

table\_name1 join\_type JOIN table\_name2 ON (join\_condition);

**Types of SQL Joins:**

* Inner Join(Equi join)
* Outer Join
* Left Outer Join
* Right Outer Join
* Full Outer Join
* Self-join
* Non-equi join
* Natural join
* **Inner join:** The INNER JOIN is the most common [type of join](https://www.tutorialrepublic.com/sql-tutorial/sql-joining-tables.php#types-of-joins). It returns only those rows that are a match (common) in both joined tables. i.e. it gives Intersection of two tables.

**Syntax:** SELECT col1, col2 FROM table1 INNER JOIN table2 ON table1.column\_name = table2.column\_name;

**emp dept**

|  |  |  |  |
| --- | --- | --- | --- |
| **emp\_id** | **emp\_name** | **hire\_date** | **dept\_id** |
| **1** | **Eshan** | **01-May-2001** | **4** |
| **2** | **Tony** | **15-July-2002** | **1** |
| **3** | **Sara** | **18-Oct-2005** | **5** |
| **4** | **Ravi** | **03-Jan-2007** | **3** |
| **5** | **Mahesh** | **24-June-2008** | **null** |

|  |  |
| --- | --- |
| **dept\_id** | **dept\_name** |
| 1 | Administration |
| 2 | Customer Service |
| 3 | Finance |
| 4 | HR |
| 5 | Sales |

Retrieves the employee's id, name, hiring date and their department by joining the ‘emp’ and ‘dept’tables together using the common ‘dept\_id’ column. It removes those employees who are not assigned to any department.

* **On Clause:** SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp e

**INNER JOIN** dept d **ON** e.dept\_id=d.dept\_id ORDER BY emp\_id;

* **Using Clause:** SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp e

**JOIN** dept d USING(dept\_id);

* **Where Clause:** SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp

e **,** dept d **where** e.dept\_id=d.dept\_id ORDER BY emp\_id;

|  |  |  |  |
| --- | --- | --- | --- |
| **emp\_id** | **emp\_name** | **hire\_date** | **dept\_name** |
| **1** | **Eshan** | **01-May-2001** | **HR** |
| **2** | **Tony** | **15-July-2002** | Administration |
| **3** | **Sara** | **18-Oct-2005** | Sales |
| **4** | **Ravi** | **03-Jan-2007** | Finance |

The result set contains only those employees whose *‘*dept\_id’ value is present and that value also exists in the ‘dept\_id’ column of the dept table.

Equality operator (=) is known as EQUI JOIN. An inner join connects two tables on a

column with the same data type. Only the rows where the column values match are returned; unmatched rows are not returned.

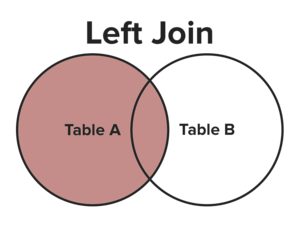
* **Non-Equi Join :** When a joining condition contains any operator other than equality

operator like >, <, >=, <=, then the join is known as NON-EQUI JOIN.

Example: SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp

e **,** dept d **where** e.dept\_id > d.dept\_id ORDER BY emp\_id;

* **Outer join:** Outer join returns all matching rows and non-matching rows from both tables. The outer join operator in Oracle (+) is used on one side of the join condition only.
* **Left outer join:** A LEFT JOIN return all rows from the “left” table, and matching rows from the “right” table.  If there are no matches in the right table, return Null values for those columns.



**Syntax:** SELECT column\_names FROM table\_name1 **LEFT JOIN** table\_name2 **ON**

table\_name1.column\_name=table\_name2.column\_name;

**OR**

SELECT column\_names FROM table\_name1, table\_name2 WHERE

table\_name1.column\_name (+) = table\_name2.column\_name;

**SQL outer join operator in Oracle (+) is used on left side of the join condition.**

**Example:** SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp e

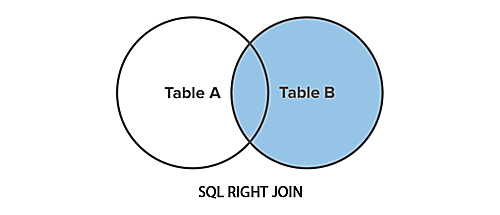
**LEFT JOIN** dept d **ON** e.dept\_id=d.dept\_id ORDER BY emp\_id;

SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp e, dept d

WHERE e.dept\_id=d.dept\_id(+);

|  |  |  |  |
| --- | --- | --- | --- |
| **emp\_id** | **emp\_name** | **hire\_date** | **dept\_name** |
| **1** | **Eshan** | **01-May-2001** | **HR** |
| **2** | **Tony** | **15-July-2002** | Administration |
| **3** | **Sara** | **18-Oct-2005** | Sales |
| **4** | **Ravi** | **03-Jan-2007** | Finance |
| **5** | **Mahesh** | **24-June-2008** | **null** |

 If there is a row in the left table but no match in the right table, then the associated result row contains NULL values

* **Right outer join:** A right outer join return all rows from the “right” table, and matching rows from the “left” table. If there are no matches in the left table, return Null values for those columns.

**Syntax:** SELECT column\_names FROM table\_name1 **RIGHT JOIN** table\_name2 **ON** table\_name1.column\_name=table\_name2.column\_name;

**OR**

SELECT column\_names FROM table\_name1, table\_name2 WHERE

table\_name1.column\_name = table\_name2.column\_name(+);

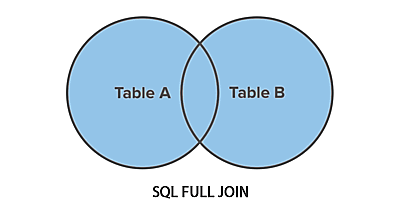
**SQL outer join operator in Oracle (+) is used on left side of the join condition***.*

**Example:** SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp e

**RIGHT JOIN** dept d **ON** e.dept\_id=d.dept\_id ORDER BY emp\_name;

|  |  |  |  |
| --- | --- | --- | --- |
| **emp\_id** | **emp\_name** | **hire\_date** | **dept\_name** |
| **2** | **Tony** | **15-July-2002** | Administration |
| **NULL** | **NULL** | **NULL** | Customer Service |
| **4** | **Ravi** | **03-Jan-2007** | Finance |
| **3** | **Sara** | **18-Oct-2005** | Sales |

Retrieve the names of all departments as well as the details of employees who're working in that department. But in some department no employee is working currently.

* **Full Join:** This will display the all matching and the non-matching records from both tables. SQL full outer join is used to combine the result of both left and right outer join and returns all rows it’s matched or unmatched from the both tables.

**Syntax:** SELECT column\_names FROM table\_name1 **FULL JOIN** table\_name2 **ON** table\_name1.column\_name=table\_name2.column\_name;

**Example:** SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp e

**FULL JOIN** dept d **ON** e.dept\_id=d.dept\_id ORDER BY emp\_name;

Some databases, such as Oracle, MySQL do not support full joins. In that case you can use the UNION ALL operator to combine the LEFT JOIN and RIGHT JOIN as follows:

SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp e **LEFT JOIN** dept d **ON** e.dept\_id = d.dept\_id **UNION ALL** SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp e **RIGHT JOIN** dept d **ON** e.dept\_id = d.dept\_id ORDER BY emp\_name;

|  |  |  |  |
| --- | --- | --- | --- |
| **emp\_id** | **emp\_name** | **hire\_date** | **dept\_name** |
| **NULL** | **NULL** | **NULL** | Customer Service |
| **1** | **Eshan** | **01-May-2001** | **HR** |
| **1** | **Eshan** | **01-May-2001** | **HR** |
| **5** | **Mahesh** | **24-June-2008** | **null** |
| **4** | **Ravi** | **03-Jan-2007** | Finance |
| **4** | **Ravi** | **03-Jan-2007** | Finance |
| **3** | **Sara** | **18-Oct-2005** | Sales |
| **3** | **Sara** | **18-Oct-2005** | Sales |
| **2** | **Tony** | **15-July-2002** | Administration |
| **2** | **Tony** | **15-July-2002** | Administration |

* **Self join :** Self-Join is a specific type of Join. In Self Join, a table is joined with itself (Unary relationship). A self-join simply specifies that each rows of a table is

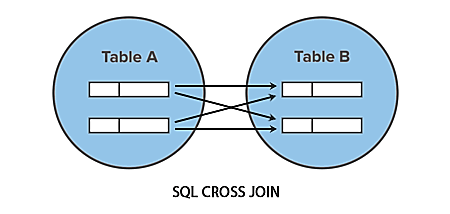
combined with itself and every other row of the table.

SELECT  a.emp\_name, b.dept\_id, a.emp\_id  FROM emp a, emp b  WHERE a.emp\_id > emp\_id;

* **Natural join:** The natural join is a type of Equi Join and is structured in such a way that, columns with same name of associated tables will appear once only.

**Syntax:** SELECT \* FROM table1 NATURAL JOIN table2;

SELECT \* FROM emp **NATURAL JOIN** dept; or SELECT \* FROM emp, dept;

* **Cross Join (Cartesian Products):** If you don't specify a join condition when joining two tables, database system combines each row from the first table with each row from the second table. This type of join is called a cross join or a Cartesian product.

If there are "x" rows in table1 and "y" rows in table2 then the cross join result set have x\*y rows. It normally happens when no matching join columns are specified. If a join condition is not specified, Oracle performs a Cartesian product.

**Syntax:** SELECT \* FROM [TABLE1] **CROSS JOIN** [TABLE2];

**OR**

SELECT \* FROM [TABLE\_NAME1], [TABLE\_NAME2] ;

**Example:** SELECT e.emp\_id, e.emp\_name, e.hire\_date, d.dept\_name FROM emp e

**CROSS JOIN** dept d;

**OR**

SELECT \* FROM emp,dept;

* **Relational Decomposition**
* When a relation in the relational model is not in appropriate normal form then the decomposition of a relation is required.
* Decomposition is the process of dividing given table (relation R) into two or more tables (relation R1,R2).
* If the relation has no proper decomposition, then it may lead to problems like loss of information.
* Decomposition is used to eliminate some of the problems of bad design like anomalies, inconsistencies, and redundancy.
* **Lossy join:**
* "The decomposition of relation R into R1 and R2 is **lossy** when the join of R1 and R2 does not produce the original relation as in R." This is Lossy join decomposition is also known as **careless decomposition.**
* One of the disadvantages some information is lost during retrieval of original relation or table.

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **Sname** | **Dept** |
| 111 | parimal | COMPUTER |
| 222 | parimal | ELECTRICAL |

* **Example: Student Table**

This relation is decomposed into two relation no\_name and name\_dept:

create table **Stu\_name** as select roll\_no, sname from student;

create table **Name\_dept** as select sname, dept from student;

**Stu\_name                                             Name\_dept**

|  |  |
| --- | --- |
| **Roll\_no** | **Sname** |
| 111 | parimal |
| 222 | parimal |

|  |  |
| --- | --- |
| **Sname** | **Dept** |
| parimal | COMPUTER |
| parimal | ELECTRICAL |

In lossy decomposition, Extra tuples are generated when a natural join is applied to the relations in the decomposition.

select \* from Stu\_name natural join Name\_dept;

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **Sname** | **Dept** |
| 111 | parimal | COMPUTER |
| 111 | parimal | ELECTRICAL |
| 222 | parimal | COMPUTER |
| 222 | parimal | ELECTRICAL |

Extra Tuples  
The above decomposition is a bad decomposition or Lossy decomposition. Extra tuples make the identification of the original tuples difficult.

* **Lossless Join:**
* "The decomposition of relation R into sub relations R1, R2……Rn is lossless when the

join of the sub relations results in the same relation R". This decomposition is called lossless join decomposition.

|  |
| --- |
| **R1 ⋈ R2 ⋈ R3 ……. ⋈ Rn = R** Where **⋈** is a natural join operator |

* A relational table is decomposed into two or more smaller tables, in such a way the designer can capture the accurate content of the original table by joining the decomposed parts. This is called lossless-join or non-additive join decomposition.

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **Sname** | **Dept** |
| 111 | parimal | COMPUTER |
| 222 | parimal | ELECTRICAL |

* Consider **STUDENT** table with three attribute roll\_no , sname and department.

This relation is decomposed into two relation Stu\_name and Stu\_dept:

create table **Stu\_name** as select roll\_no, sname from student;

create table **Stu\_dept** as select roll\_no, dept from student;

|  |  |
| --- | --- |
| **Roll\_no** | **Dept** |
| 111 | COMPUTER |
| 222 | ELECTRICAL |

|  |  |
| --- | --- |
| **Roll\_no** | **Sname** |
| 111 | parimal |
| 222 | parimal |

**Stu\_name table                                                        Stu\_dept table**

Now, when these two relations are joined on the common column 'roll\_no', the resultant relation will look like

select \* from Stu\_name natural join Stu\_dept;

|  |  |  |
| --- | --- | --- |
| **Roll\_no** | **Sname** | **Dept** |
| 111 | parimal | COMPUTER |
| 222 | parimal | ELECTRICAL |

In lossless decomposition, no any extra tuples are generated when a natural joined is applied to the relations in the decomposition.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Q1. What are the advantages of DBMS? List different types of database users.

Q2. Define Schema and Instances. Distinguish between primary and super key.

Q3. Explain Dr. Codd’s rule.

Q4. What is Join? Explain types of join with example.

Q5. Why 3-tier architecture useful? Explain Relational model.