Primary Key

A primary key is a unique identifier for a record in a database table.

Syntax for creating primary key:

```
Create table student123
(
student_id int unique,
student_name varchar(30),
student_age int check (student_age>0)
)
```

Adding primary key constraints using Alter

```
Alter table student1
Add constraint pk_student_idprimary key (student_id)
```

if not null is not active on the column on which you are trying to make the primary key using the alter command first you have to make that column not null.

```
Alter table student1
Alter column student_id int not null
```

Foreign key

A foreign key is a column or a set of columns in a database table that refers to the primary key of another table, establishing a link between the two tables.

```
Create table student_course
(
student_id int foreign keyreferences student(student_id),
student_course varchar(100),
professor varchar(100)
)
```

Adding foreign key constraints using Alter

```
Alter table student_course
Add constraint fk_student_Id foreign key  references
student(student_id)
```

Unique Key

A unique constraint ensures that all values in a specified column or combination of columns are distinct within a database table.

```
Create table student123
(
student_id int unique,
student_name varchar(30),
student_age int check (student_age>0)
)
```

Adding unique constraints using Alter

```
ALTER TABLE student
ADD CONSTRAINT uk_studentid UNIQUE (student_id)
```

Constraints, Joins, Set Operators

Default

Use to give a default value for a column if the user not giving any value

```
Create table student_course
(
student_id int foreign key references student(student_id),
student_course varchar(100),
Professor Varchar(100) Default 'to be decided'
)
```

Adding default constraints using Alter

```
Alter table student_course
Add default 'to be decided' for professor
```

Check Constraint

Use to give a default value for a column if the user not giving any value

```
Create table student
(
student_id int primary key,
student_name varchar(30),
student_age int check (student_age>0)
)
```

Adding check constraints using Alter

```
Alter table student
Add constraint ck_age check(student_age>0)
```

Joins

Sample Tables

employee_id	employee_name	department_id	salary
1	John Doe	1	60000.00
2	Jane Smith	2	55000.00
3	Bob Johnson	1	70000.00
4	Alice Brown	3	50000.00
5	Charlie Wilson	2	65000.00
6	Sachin	NULL	60000.00

Department

department_id	department_name
1	IT
2	HR
3	Finance
4	Admin

Syntax

Inner join

```
SELECT *
FROM employees
INNER JOIN departments
ON
employees.department_id =
departments.department_id;
```

employee_id	employee_name	department_id	salary	department_id	department_name
1	John Doe	1	60000.00	1	IT
2	Jane Smith	2	55000.00	2	HR
3	Bob Johnson	1	70000.00	1	IT
4	Alice Brown	3	50000.00	3	Finance
5	Charlie Wilson	2	65000.00	2	HR

Left join


```
SELECT *
FROM employees
LEFT JOIN departments
ON
employees.department_id =
departments.department_id;
```

employee_id	employee_name	department_id	salary	department_id	department_name
1	John Doe	1	60000.00	1	IT
2	Jane Smith	2	55000.00	2	HR
3	Bob Johnson	1	70000.00	1	IT
4	Alice Brown	3	50000.00	3	Finance
5	Charlie Wilson	2	65000.00	2	HR
6	Sachin	NULL	60000.00	NULL	NULL

Right join

```
SELECT *
FROM employees
RIGHT JOIN departments
ON
employees.department_id =
departments.department_id;
```

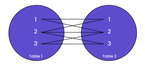
employee_id	employee_name	department_id	salary	department_id	department_name
1	John Doe	1	60000.00	1	IT
3	Bob Johnson	1	70000.00	1	IT
2	Jane Smith	2	55000.00	2	HR
5	Charlie Wilson	2	65000.00	2	HR
4	Alice Brown	3	50000.00	3	Finance
NULL	NULL	NULL	NULL	4	Admin

Full join

```
SELECT *
FROM employees
FULL JOIN departments
ON
employees.department_id =
departments.department_id;
```

employee_id	employee_name	department_id	salary	department_id	department_name
1	John Doe	1	60000.00	1	IT
2	Jane Smith	2	55000.00	2	HR
3	Bob Johnson	1	70000.00	1	IT
4	Alice Brown	3	50000.00	3	Finance
5	Charlie Wilson	2	65000.00	2	HR
6	Sachin	NULL	60000.00	NULL	NULL
NULL	NULL	NULL	NULL	4	Admin

Cross join



```
SELECT *
FROM employees
CROSS JOIN departments;
```

employee_id	employee_name	department_id	salary	department_id	department_name
1	John Doe	1	60000.00	1	IT
2	Jane Smith	2	55000.00	1	IT
3	Bob Johnson	1	70000.00	1	IT
4	Alice Brown	3	50000.00	1	IT
5	Charlie Wilson	2	65000.00	1	IT
6	Sachin	NULL	60000.00	1	IT
1	John Doe	1	60000.00	2	HR
2	Jane Smith	2	55000.00	2	HR
3	Bob Johnson	1	70000.00	2	HR
4	Alice Brown	3	50000.00	2	HR
5	Charlie Wilson	2	65000.00	2	HR
6	Sachin	NULL	60000.00	2	HR
1	John Doe	1	60000.00	3	Finance
2	Jane Smith	2	55000.00	3	Finance
3	Bob Johnson	1	70000.00	3	Finance
4	Alice Brown	3	50000.00	3	Finance
5	Charlie Wilson	2	65000.00	3	Finance
6	Sachin	NULL	60000.00	3	Finance
1	John Doe	1	60000.00	4	Admin
2	Jane Smith	2	55000.00	4	Admin
3	Bob Johnson	1	70000.00	4	Admin
4	Alice Brown	3	50000.00	4	Admin
5	Charlie Wilson	2	65000.00	4	Admin
6	Sachin	NULL	60000.00	4	Admin

Self join

```
SELECT *
FROM employees e1
INNER JOIN employees e2
ON e1.department_id =
e2.department_id;
```

employee_id	employee_name	department_id	salary	employee_id	employee_name	department_id	salary
1	John Doe	1	60000.00	1	John Doe	1	60000.00
3	Bob Johnson	1	70000.00	1	John Doe	1	60000.00
2	Jane Smith	2	55000.00	2	Jane Smith	2	55000.00
5	Charlie Wilson	2	65000.00	2	Jane Smith	2	55000.00
1	John Doe	1	60000.00	3	Bob Johnson	1	70000.00
3	Bob Johnson	1	70000.00	3	Bob Johnson	1	70000.00
4	Alice Brown	3	50000.00	4	Alice Brown	3	50000.00
2	Jane Smith	2	55000.00	5	Charlie Wilson	2	65000.00
5	Charlie Wilson	2	65000.00	5	Charlie Wilson	2	65000.00

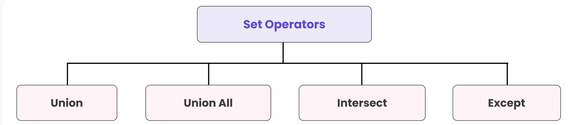
Set Operators

Employee_Details1

Emp_Id	Emp_Name	Emp_Salary
1	Mary	25000
2	Ramesh	45000
3	Lily	35000

Employee_Details2

Emp_Id	Emp_Name	Emp_Salary
4	Jack	75000
5	Rose	47000
3	Lily	35000



Set operators are used to combine two result set together

Union Output

Get all distinct rows between **Employee_Details1** and **Employee_Details2 table.**

Emp_Id	Emp_Name	Emp_Salary
1	Mary	25000
2	Ramesh	45000
3	Lily	35000
4	Jack	75000
5	Rose	47000

```
SELECT * FROM Employee_Details1
UNION
SELECT * FROM Employee_Details2;
```

UNION: Combines the result sets of two SELECT queries, removing duplicate rows.

Union All Output

Get all rows with duplicates between **Employee_Details1** and **Employee_Details2 table.**

Emp_Id	Emp_Name	Emp_Salary
1	Mary	25000
2	Ramesh	45000
3	Lily	35000
4	Jack	75000
5	Rose	47000
3	Lily	35000

```
SELECT * FROM Employee_Details1
UNION ALL
SELECT * FROM Employee_Details2;
```

UNION ALL: Combines the result sets of two SELECT queries, including all rows, including duplicates.

Intersect Output

Get common rows between **Employee_Details1** and **Employee_Details2 table**

Emp_Id	Emp_Name	Emp_Salary
3	Lily	35000

INTERSECT: Returns common rows between the result sets of two SELECT queries, removing duplicates

```
SELECT * FROM Employee_Details1
INTERSECT
SELECT * FROM Employee_Details2;
```

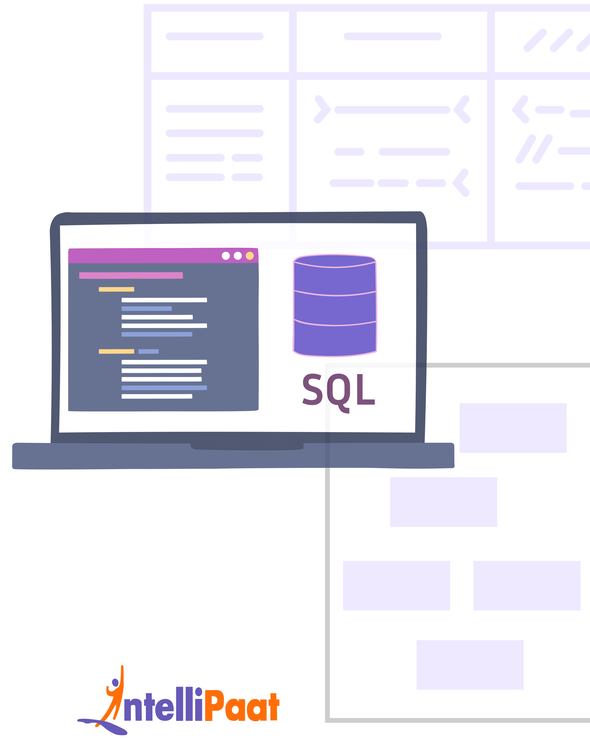
Except Output

Get the rows from **Employee_Details1** except the rows that are present in **Employee_Details2 table.**

Emp_Id	Emp_Name	Emp_Salary
1	Mary	25000
2	Ramesh	45000

Except: Returns distinct rows from the result of the first SELECT query that are not present in the result of the second SELECT query.

```
SELECT * FROM Employee_Details1
EXCEPT
SELECT * FROM Employee_Details2;
```



IntelliPaat

SQL
Cheat Sheet