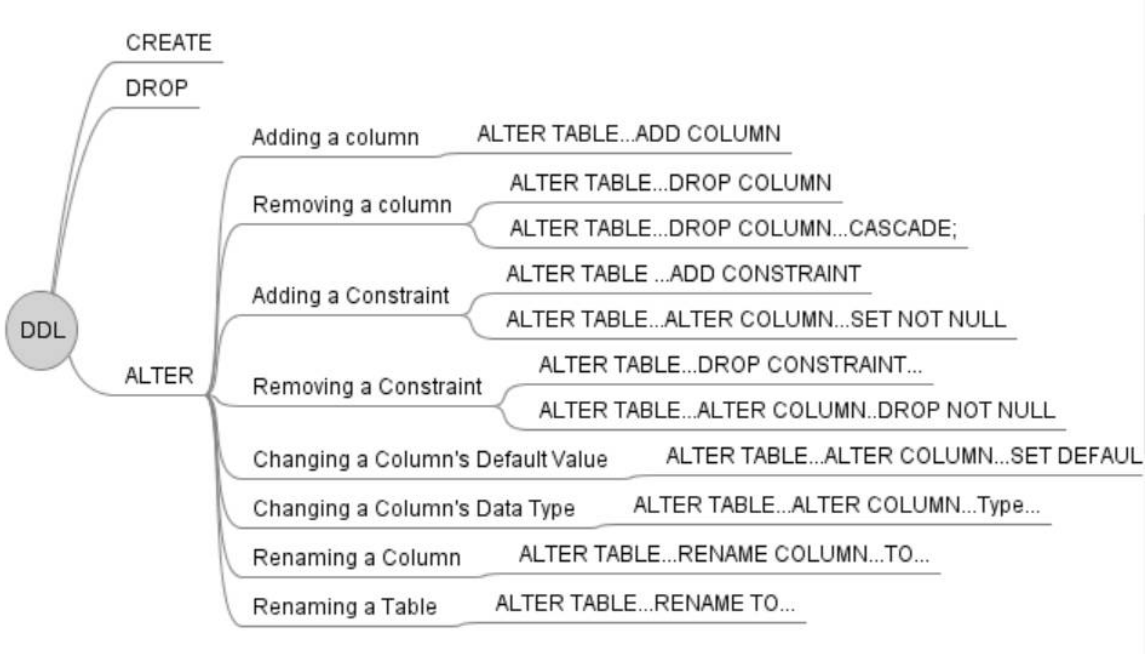
DDL

In a relational database, the raw data is stored in tables, so the majority of this chapter is devoted to explaining how tables are created and modified and what features are available to control what data is stored in the tables

> Data definition language: **drop, create, alter**



Here we use following tables for example

```
create table dept(
id int primary key,
name varchar(40)
);

insert into dept values(1, '开发部');
insert into dept values(2, '测试部');

create table emp(
id int primary key,
name varchar(30),
salary numeric(9,2),
deptId int references dept(id)
);
```

Drop

Some additions to drop

There are several caveats to be aware of when using this option. Only one index name can be specified, and the `CASCADE` option is not supported. (Thus, an index that supports a `UNIQUE` or `PRIMARY KEY` constraint cannot be dropped this way.) Also, regular `DROP INDEX` commands can be performed within a transaction block, but `DROP INDEX CONCURRENTLY` cannot. Lastly, indexes on partitioned tables cannot be dropped using this option.

Alter

Some usage scenes:

ALTER TABLE -- change the definition of a table https://www.postgresql.org/docs/16/sql-altertable.html

ALTER FOREIGN TABLE -- change the definition of a foreign table https://www.postgresql.org/docs/16/sql-alterforeigntable.html

ALTER ROLE -- change a database role https://www.postgresql.org/docs/16/sql-alterrole.html

ALTER FUNCTION -- change the definition of a function https://www.postgresql.org/docs/16/sql-alterfunction.html

ALTER PROCEDURE -- change the definition of a procedure https://www.postgresql.org/docs/16/sql-alterprocedure.html

ALTER DEFAULT PRIVILEGES -- define default access privileges https://www.postgresql.org/docs/16/sql-alterdefaultprivileges.html

ALTER INDEX -- change the definition of an index https://www.postgresql.org/docs/16/sql-alterindex.html

Experiment 1: Adding a column

> *ALTER TABLE table_name ADD COLUMN description text;*

```
ALTER TABLE emp ADD column phone varchar(30);
ALTER TABLE emp ADD column office varchar(30) check (office <> '');
```

Tips: The new column is initially filled with whatever default value is given (null if you don't specify a `DEFAULT` clause).

Experiment 2-1: Removing a column

> *ALTER TABLE table_name DROP COLUMN description;*

```
ALTER TABLE emp DROP column phone;

ALTER TABLE emp DROP column deptId;
```

Tips: Whatever data was in the column disappears. Table constraints involving the column are dropped, too. However, if the column is referenced by a foreign key constraint of another table, PostgreSQL will not silently drop that constraint.

Experiment 2-2: Removing a column using CASCADE

> *ALTER TABLE table_name DROP COLUMN description CASCADE;*

```
ALTER TABLE emp DROP column deptId CASCADE;
```

Tips: You can authorize dropping everything that depends on the column by adding `CASCADE`:

Experiment 3: Adding a Constraint

> *ALTER TABLE table_name ADD CHECK (name <> '');*
> *ALTER TABLE table_name ADD CONSTRAINT some_name UNIQUE (product_no);*
> *ALTER TABLE table_name ADD FOREIGN KEY (fk) REFERENCES product_groups;*
>
> ......

```
ALTER TABLE dept ADD PRIMARY KEY(id);
ALTER TABLE emp ADD FOREIGN KEY (deptId) REFERENCES dept(id);
ALTER TABLE emp ADD CONSTRAINT phone_unique UNIQUE (phone);
ALTER TABLE dept ADD COLUMN  id int;
```

Tips: To add a not-null constraint, which cannot be written as a table constraint, use this syntax:

> *ALTER TABLE table_name ALTER COLUMN col_name SET NOT NULL;*

```
ALTER TABLE emp ALTER COLUMN name SET NOT NULL;
```

Experiment 4-1: Removing a Constraint

> *ALTER TABLE table_name DROP CONSTRAINT some_name;*

```
ALTER TABLE emp DROP CONSTRAINT some_name;-- some_name key name
```

Experiment 4-2: Removing a Constraint Not Null

> *ALTER TABLE table_name ALTER COLUMN col_name DROP NOT NULL;*

```
ALTER TABLE emp ALTER COLUMN name DROP NOT NULL;
```

Tips: Not-null constraints do not have names.

Experiment 5: Changing a Column's Default Value

> *ALTER TABLE table_name ALTER COLUMN col_name SET DEFAULT val;*

```
ALTER TABLE emp ALTER COLUMN salary SET DEFAULT 0.00;
ALTER TABLE emp ALTER COLUMN salary DROP DEFAULT;
```

Tips: Note that this doesn't affect any existing rows in the table, it just changes the default for future `INSERT` commands.

Experiment 6:

Changing a Column's Data Type

> *ALTER TABLE table_name ALTER COLUMN col_name Type col_type;*

```
ALTER TABLE emp ALTER COLUMN salary TYPE numeric(9,4);
```

Experiment 7: Renaming a Column

> *ALTER TABLE table_name RENAME COLUMN old_name TO new_name;*

```
ALTER TABLE emp RENAME COLUMN phone TO Tel_No;
```

Experiment 8: Renaming a Table

> *ALTER TABLE table_old RENAME TO table_new;*

```
ALTER TABLE emp RENAME TO employee;
```

DML

> Data manipulation language: **insert, update, delete**

Using the following table

```
drop table emp;
drop table dept;

create table dept(
id int primary key,
name varchar(40)
);

create table emp(
id int primary key,
```

```
name varchar(30),
salary numeric(9,2),
deptId int references dept(id)
);
```

Experiment 9: Insert— create new rows in a table

> *INSERT INTO table(...) VALUES(..);*

https://www.postgresql.org/docs/current/sql-insert.html

```
INSERT INTO dept(id, name) values(1, '开发部');
INSERT INTO dept values('测试部',2);
INSERT INTO dept(id, name) values
            (3, '财务部'),
            (4, '销售部'); \--insert multi-records
```

Tips: When inserting a lot of data at the same time, consider using the COPY command. It is not as flexible as the INSERT command, but is more efficient.

Experiment 10: Update— update rows of a table

> *UPDATE table SET ...;*

https://www.postgresql.org/docs/current/sql-update.html

```
UPDATE dept SET name='市场部' where id=4;
```

Tips: To update existing rows, use the UPDATE command. This requires three pieces of information:

1. The name of the table and column to update
2. The new value of the column
3. Which row(s) to update

Experiment 11:

Delete— delete rows of a table

> *DELETE FROM table WHERE ...;*

https://www.postgresql.org/docs/current/sql-delete.html

```
DELETE FROM dept WHERE name='市场部';
DELETE FROM dept;
```

Tips: You can also remove groups of rows matching a condition, or you can remove all rows in the table at once

DQL

Here we use shenzhen_metro.sql, please download first

The general syntax:

> *[WITH with_queries] SELECT select_list FROM table_expression [sort_specification]*

https://www.postgresql.org/docs/current/sql-select.html

Experiment 12:

Select

```
create table table1(
    a integer,
    b integer,
    c integer
);
insert into table1 values(1,2,3);

SELECT * FROM table1;
SELECT a, b + c FROM table1;
SELECT 3 * 4;-- as a calculator
SELECT random();
```

Tips: "*" present for all attribute or any attribute.

Experiment 13:

Select...(as)...from...where ... or, and, not

> *WHERE condition*

```
select station_id , english_name , chinese_name
from stations
where district = 'Nanshan' or district ='Bao''an';

select station_id as sid, english_name as e_n, chinese_name c_n
from stations
where district = 'Nanshan' and latitude >22.54;

select station_id, english_name, chinese_name
from stations
where not(district = 'Nanshan' or district ='Bao''an');
```

Tips: where `condition` is any expression that evaluates to a result of type `boolean`. Any row that does not satisfy this condition will be eliminated from the output. A row satisfies the condition if it returns true when the actual row values are substituted for any variable references.

Experiment 14:

>,<,>=,<=,!= or <>

https://www.postgresql.org/docs/13/sql-syntax-lexical.html#SQL-SYNTAX-IDENTIFIERS Table 4.2 **Operator Precedence**

Table Operator Precedence (highest to lowest)

Operator/Element	Associativity	Description
.	left	table/column name separator
::	left	PostgreSQL-style typecast
[ ]	left	array element selection
+ -	right	unary plus, unary minus
COLLATE	left	collation selection
AT	left	AT TIME ZONE
^	left	exponentiation
* / %	left	multiplication, division, modulo
+ -	left	addition, subtraction
(any other operator)	left	all other native and user-defined operators
BETWEEN IN LIKE ILIKE SIMILAR		range containment, set membership, string matching
< > = <= >= <>		comparison operators
IS ISNULL NOTNULL		IS TRUE, IS FALSE, IS NULL, IS DISTINCT FROM, etc
NOT	right	logical negation
AND	left	logical conjunction
OR	left	logical disjunction

Experiment 15: in, not in

```
select station_id, english_name, chinese_name
from stations
where district in ('Nanshan' ,'Bao''an');

select station_id, english_name, chinese_name
from stations
where district not in ('Nanshan' ,'Bao''an');
```

Experiment 16: like ,%, _

```
select station_id, english_name,chinese_name
from stations
where chinese_name like '%湾%';

select station_id, english_name,chinese_name
from stations
where chinese_name like '湾%';

select station_id, english_name,chinese_name
from stations
where chinese_name like '__湾';
```

Experiment 17: NULL, NOT NULL

```
select station_id, english_name,chinese_name
from stations
where latitude is NULL;

select station_id, english_name,chinese_name
from stations
where latitude is not NULL;
```

Tips: "NULL" is not a value, so we use "is" connect with NULL/NOT NULL.

deal with the select data

Experiment 18: ||

```
select '南山区地铁站平均经纬度为:'||avg(latitude)||'::'||avg(longitude)
from stations
where latitude is not NULL and longitude is not null and district='Nanshan';
```

Experiment 19: as

```
select '南山区地铁站平均经纬度为:'||avg(latitude)||'::'||avg(longitude) as 输出信息
from stations
where latitude is not NULL and longitude is not null and district='Nanshan';
```

Experiment 20: case...when...then...else...end

```
select district,avg(longitude) ,
case
when avg(longitude)>114
then 'North'
else 'West'
end as sign
from stations
where longitude is not null
group by district;
```

some other functions

- upper,lower
- trim:trim(' Oops ') return 'Oops'
- substr:substr('Nanshan', 3, 2) return 'ns'
- replace:replace('Sheep', 'ee', 'i') return 'Ship'
- length
- round, trunc

Subquery

Experiment 21: exist

```
select station_id, english_name,chinese_name from stations where exists
(select district from stations where district='Nanshan');

select station_id, english_name,chinese_name from stations where exists
(select district from stations where district='Nansha');
```

Experiment 22: in

```
select station_id, english_name,chinese_name,district from stations where
district in
(select distinct (district)from stations where latitude>22.6);
```

Tips: you may see the following usage in practical application

> *SELECT ... FROM table WHERE c1 > 5*
>
> *SELECT ... FROM table WHERE c1 IN (1, 2, 3)*
>
> *SELECT ... FROM table WHERE c1 IN (SELECT c1 FROM t2)*
>
> *SELECT ... FROM table WHERE c1 IN (SELECT c3 FROM t2 WHERE c2 = table.c1 + 10)*

> *SELECT ... FROM table WHERE c1 BETWEEN (SELECT c3 FROM t2 WHERE c2 = table.c1 + 10) AND 100*
>
> *SELECT ... FROM table WHERE EXISTS (SELECT c1 FROM t2 WHERE c2 > table.c1)*

Aggregate Functions

Aggregate functions compute a single result from a set of input values. The built-in general-purpose aggregate functions are listed in https://www.postgresql.org/docs/13/functions-aggregate.html Table 9.55

Experiment 23: count

```
select count(*)
from stations
where latitude is NULL;
```

Tips: Computes the number of input rows in which the input value is not null.

Experiment 24: max, min, avg

```
select avg(latitude), min(latitude), max(latitude)
from stations
where latitude is not NULL;
```

Experiment 25: group by

```
select count(district),district
from stations
where district <> ''
group by district;
```

Tips: `GROUP BY` will condense into a single row all selected rows that share the same values for the grouped expressions.

https://www.postgresql.org/docs/13/sql-select.html#SQL-GROUPBY

Experiment 26: having

```
select count(district),district
from stations
group by district
having district<>'';
```

Tips: After passing the `WHERE` filter, the derived input table might be subject to grouping, using the `GROUP BY` clause, and elimination of group rows using the `HAVING` clause.

https://www.postgresql.org/docs/13/sql-select.html#SQL-HAVING

Experiment 27: distinct

```
select count(distinct (district))
from stations
where latitude>22.6;
```

Experiment 28: order by...asc/dec

```
select station_id, english_name,chinese_name,latitude
from stations
order by latitude asc;--default is asc

select station_id, english_name,chinese_name,latitude
from stations
order by latitude desc;

select station_id, english_name,chinese_name,latitude
from stations
order by latitude desc nulls last ;
```

Tips: null, order by...desc/asc...nulls last/first

Experiment 29: limit , offset

```
select count(district),district
from stations
where district <> ''
group by district
order by count(district) limit 3;

select count(district),district
from stations
where district <> ''
group by district
order by count(district) limit 3 offset 2;
```

Date

There are so many functions about time/date, we only pick some example in common use. More details in https://www.postgresql.org/docs/16/functions-datetime.html

Experiment 30: the present date and time

```
select now();

select current_date||' '||current_time;
```

Tips: Here the default time zone is UTC other than Beijing time.

```
show timezone;
```

Experiment 31: date/time operator

1. date+int-->date

```
select current_date+7;
select date'2024-9-1'+7;
```

2. time/timestamp+interval-->time/timestamp

```
select time '08:00' + interval '2 hours';
select timestamp'2024-09-10 08:00' + interval '2 hours';
```

Tips: More detail in link https://www.postgresql.org/docs/16/functions-datetime.html

Experiment 32: extract field

> *EXTRCT(field FROM source)*

```
select extract(year from current_date);
select extract(minute from current_timestamp);
```

Tips: valid field names, year, month, century, decade, hour,minute...