

SQL，或结构化查询语言，是一种与数据库对话的语言。它允许您选择特定的数据并构建复杂的报告。今天，**SQL** 是一种通用的数据语言。它几乎被用于所有处理数据的技术中。

示例数据

COUNTRY				
id	name	population	area	
1	France	66600000	640680	
2	Germany	80700000	357000	
...	
CITY				
id	name	country_id	population	rating
1	Paris	1	2243000	5
2	Berlin	2	3460000	3
...

查询单个表

从国家表格中获取所有列：
SELECT *
FROM country;

从城市表中获取ID列和名称列：
SELECT id, name
FROM city;

获取按照评级列升序排列的城市名称：
SELECT name
FROM city
ORDER BY rating **[ASC]**;

获取按照评级列降序排列的城市名，
SELECT name
FROM city
ORDER BY rating **DESC**;

ALIASES 别名

COLUMNS

SELECT name **AS** city_name
FROM city;

TABLES

SELECT co.name, ci.name
FROM city **AS** ci
JOIN country **AS** co
ON ci.country_id = co.id;

过滤输出

COMPARISO OPERATORS 比较运算符

获取评级在三级以上的城市：
SELECT name
FROM city
WHERE rating > 3;

获取不是柏林也不是马德里的城市名称：
SELECT name
FROM city
WHERE name != 'Berlin'
AND name != 'Madrid';

TEXT OPERATORS

获取以p开头或者以s结尾的城市的名称：
SELECT name
FROM city
WHERE name **LIKE** 'P%'
OR name **LIKE** '%s';

获取以字母“ ublin ”开头的城市名称（比如爱尔兰的都柏林或波兰的都柏林）：
SELECT name
FROM city
WHERE name **LIKE** '_ublin';

OTHER OPERATORS

获取人口在50万至500万之间的城市的名称：
SELECT name
FROM city
WHERE population **BETWEEN** 500000 **AND** 5000000;

获取评级值不为null的城市的名称：

SELECT name
FROM city
WHERE rating **IS NOT NULL**;

获取ID为1、4、7或8的国家的城市名称：
SELECT name
FROM city
WHERE country_id **IN** (1, 4, 7, 8);

查询多个表

INNER JOIN 内连接

JOIN (or explicitly **INNER JOIN**) 返回在两个表中都有匹配值的行。
SELECT city.name, country.name
FROM city
[INNER] JOIN country
ON city.country_id = country.id;

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	3	Iceland

LEFT JOIN 左连接

LEFT JOIN 返回左表中的所有行以及右表中的匹配行。如果没有匹配的，**NULLs** 将作为第二个表中的值返回。
SELECT city.name, country.name

FROM city
LEFT JOIN country
ON city.country_id = country.id;

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL

RIGHT JOIN 右连接

RIGHT JOIN 返回右表中的所有行以及左表中的匹配行。如果没有匹配的行，**NULLs** 将作为左表中的值返回。
SELECT city.name, country.name

FROM city
RIGHT JOIN country
ON city.country_id = country.id;

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
NULL	NULL	NULL	3	Iceland

FULL JOIN 全连接

FULL JOIN (or explicitly **FULL OUTER JOIN**) 返回两个表中的所有行-如果在第二个表中没有匹配的行，**NULLs** are returned.
SELECT city.name, country.name
FROM city
FULL [OUTER] JOIN country
ON city.country_id = country.id;

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL
NULL	NULL	NULL	3	Iceland

CROSS JOIN 交叉连接

CROSS JOIN 返回两个表中所有可能的行组合. 有两种语法可用
SELECT city.name, country.name
FROM city
CROSS JOIN country;

SELECT city.name, country.name
FROM city, country;

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
1	Paris	1	2	Germany
2	Berlin	2	1	France
2	Berlin	2	2	Germany

NATURAL JOIN 自然连接

NATURAL JOIN 将使用相同名称的所有列连接表。
SELECT city.name, country.name
FROM city
NATURAL JOIN country;

CITY			COUNTRY	
country_id	id	name	name	id
6	6	San Marino	San Marino	6
7	7	Vatican City	Vatican City	7
5	9	Greece	Greece	9
10	11	Monaco	Monaco	10

NATURAL JOIN 使用这些列来匹配行:
city.id, city.name, country.id, country.name.
NATURAL JOIN 在实践中很少使用。

AGGREGATION AND GROUPING聚合和分组

GROUP BY 将指定列中具有相同值的行分组在一起。它为每个唯一的值组合计算摘要(聚合)

CITY		
id	name	country_id
1	Paris	1
101	Marseille	1
102	Lyon	1
2	Berlin	2
103	Hamburg	2
104	Munich	2
3	Warsaw	4
105	Cracow	4



CITY	
country_id	count
1	3
2	3
4	2

AGGREGATE FUNCTIONS 聚合函数

- avg(expr) – 组内的行的平均值
- count(expr) – 对组内的行的值的计数
- max(expr) – 组内的最大值
- min(expr) – 组内的最小值
- sum(expr) – 组内的值的总和

EXAMPLE QUERIES 示例查询

找出城市的数量：
SELECT COUNT(*)
FROM city;

找出评级非null的城市的数量：
SELECT COUNT(rating)
FROM city;

找出不同国家值的数量：
SELECT COUNT(DISTINCT country_id)
FROM city;

找出最小和最大的国家人口：
SELECT MIN(population), MAX(population)
FROM country;

找出各自国家的城市人口总数：
SELECT country_id, SUM(population)
FROM city
GROUP BY country_id;

如果平均评级高于3.0，则找出各自国家的城市的平均评级：
SELECT country_id, AVG(rating)
FROM city
GROUP BY country_id
HAVING AVG(rating) > 3.0;

SUBQUERIES 子查询

子查询是嵌套在另一个查询中或嵌套在另一个子查询中的查询。有不同类型的子查询。

SINGLE VALUE 单一值

最简单的子查询只返回一列和一行。它可以与比较运算符=、<、<=、>或=一起使用。

此查询将查找与巴黎的评级相同的城市：

SELECT name
FROM city
WHERE rating = (
SELECT rating
FROM city
WHERE name = 'Paris'
);

MULTIPLE VALUES 多个值

一个子查询还可以返回多个列或多个行。这些子查询可以与运算符IN、EXIST S、ALL或ANY一起使用。

该查询查找人口超过2000万的国家的城市：

SELECT name
FROM city
WHERE country_id **IN** (
SELECT country_id
FROM country
WHERE population > 20000000
);

CORRELATED 关联子查询

关联子查询引用外部查询中引入的表。关联子查询依赖于外部查询。它不能独立于外部查询运行。

此查询将查找人口数量大于该国平均人口数量的城市：

SELECT *
FROM city main_city
WHERE population > (
SELECT AVG(population)
FROM city average_city
WHERE average_city.country_id = main_city.country_id
);

此查询将查找至少有一个城市的国家：

SELECT name
FROM country
WHERE **EXISTS** (
SELECT *
FROM city
WHERE country_id = country.id
);

SET OPERATIONS 集合运算符

集合操作用于将两个或多个查询的结果合并为单个结果。组合查询必须返回相同数量的列和兼容的数据类型。对应列的名称可以不同。

CYCLING			SKATING		
id	name	country	id	name	country
1	YK	DE	1	YK	DE
2	ZG	DE	2	DF	DE
3	WT	PL	3	AK	PL
...

UNION 联合

UNION 合并两个结果集的结果并删除重复项. **UNION ALL** 不删除重复的行。

这个查询显示了德国自行车手和德国滑冰手：

SELECT name
FROM cycling
WHERE country = 'DE'
UNION / UNION ALL
SELECT name
FROM skating
WHERE country = 'DE';

INTERSECT 交叉

INTERSECT 只返回两个结果集中都出现的行。

这个查询显示了同时也是德国滑冰运动员的德国自行车手：

SELECT name
FROM cycling
WHERE country = 'DE'
INTERSECT
SELECT name
FROM skating
WHERE country = 'DE';

EXCEPT 除了

EXCEPT 仅返回出现在第一个结果集中但未出现在第二个结果集中的行。

这个查询显示德国的自行车手，除非他们同时也是德国的滑冰运动员：

SELECT name
FROM cycling
WHERE country = 'DE'
EXCEPT / MINUS
SELECT name
FROM skating
WHERE country = 'DE';