SQL

Julian Eng

April 2, 2022

Contents

1 SQL

1.1 SELECT

```
SELECT <fields> FROM <table> WHERE <conditions>
```

1.2 INSERT

```
INSERT INTO <table> (field1, field2, field3) VALUES (v1, v2, v3)
```

1.3 UPDATE

```
UPDATE <table> SET f1=v1,f2=v2,f3=v3 WHERE <condition>
```

1.4 Generally…

```
SELECT column1, column2, ..., columnn
FROM table1, table2, ..., tablem
WHERE condition;
```

Joining tables by commas will give you a cartesian product.

The general form projects the given columns on the rows indicated by the condition from the cartesian product of the given tables.

1.5 DISTINCT

```
SELECT DISTINCT <columns> FROM <table> WHERE <condition>
```

The `DISTINCT` keyword will deduplicate the result of the query run without the `DISTINCT`.

1.6 NATURAL JOIN

```
SELECT a.field1, b.field1, a.field2
FROM A a, B b
WHERE a.field2 = b.field2;
```

The `FROM` clause creates a catesian product and the `WHERE` clause creates the join condition.

```
SELECT a.field1, b.field1, a.field2
FROM A AS a NATURAL JOIN B AS b;
```

Same as above. If there are no matching join fields, this is the same as cartesian product.

1.7 LEFT OUTER JOIN

```
SELECT A.field1, B.field1, B.field2
FROM A LEFT OUTER JOIN B;
```

This will join A to B on their matching column; if the column in A doesn't have a match in B, B's columns in the output will be NULL.

1.8 String Matching

```
SELECT <columns> FROM <table> WHERE field LIKE 'Foo%';
```

- `%` matches any string (regex `.*`)
- `_` matches any one character (regex `.`)

1.9 Sorting

```
SELECT <columns> FROM <table> ORDER BY field ASC;
```

(can replace `ASC` with `DESC` for reverse order)

1.10 UNION

```
<SELECT QUERY>  UNION <SELECT QUERY>
```

- All rows of the first query set union with all rows of the second query
- DUPLICATES ARE NOT PRESERVED
- If you want to preserve duplicates, use `UNION ALL`

1.11 INTERSECT

- Same as `UNION` but it's an intersection.
- `INTERSECT ALL` behaves the same way. (What happens if there's two in the first query but only one in the second query?)

1.12 EXCEPT

- Same as `UNION` but it's a set difference.
- `EXCEPT ALL` makes the output contain the difference in occurances of the rows in question.

1.13 Builtin functions

- `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`
- `SELECT COUNT(*) FROM Table;` -> create one row with the number of rows in the table.

1.14 GROUP BY

```
SELECT col, COUNT(*) as num, AVG(numcol) av
FROM Table
GROUP BY col
ORDER BY num ASC;
```

- `col` must appear in the output because it's being grouped on.
- the `COUNT` and `AVG` will act on the groups, not the entire table.

1.15 HAVING

```
SELECT col, COUNT(*) as num, AVG(numcol) av
FROM Table
GROUP BY col
HAVING num > 3
ORDER BY num ASC;
```

- `HAVING` will create a condition on the returned groups. In this case, we restrict outputs to groups with more than 3 members.

1.16 Nesting Queries

1.16.1 IN/NOT INT

```
SELECT Foo, Bar
FROM Table
WHERE Foo IN
      (SELECT Foo
       FROM Table2
       WHERE Baz = 'qux');
```

- Inner query makes a table that the outer query searches on.
- The inner query has no relation to the outer query besides providing a table to search.

1.16.2 $=, \neq, \geq, \leq, <, >$, SOME/ALL

```
SELECT Foo
FROM R, Y
WHERE R.B = Y.B AND Z > ALL
      (SELECT Z
       FROM R, Y
       WHERE R.B = Y.B AND R.A = 'Bar');
```

- Again, inner query makes a table that the outer query does something with.
- In this case, it compares each row's `Z` to **every** `Z` in the inner table.

1.16.3 Correlated Queries

```
SELECT R.E, R.Y
FROM R
WHERE NOT EXIST
      (SELECT *
       FROM U
       WHERE U.E = R.E);
```

- In this case, the inner query refers to something from the outer query (`R.E`)
- Think of it as the inner query getting evaluated for each row of the outer query