Other Useful Statements and Database Joins

SWEN-250: Personal Software Engineering

R·I·T

ROCHESTER INSTITUTE OF TECHNOLOGY

Overview

- Other Useful Statements:
  - SELECT Functions
  - SELECT Like
  - ORDER BY
  - LIMIT
  - UPDATE
  - DELETE
  - ALTER TABLE
  - DROP TABLE
- Database Info:
  - .tables
  - .schema
- Joins:
  - LEFT JOIN
  - RIGHT JOIN
  - INNER JOIN
  - FULL OUTER JOIN

Other Useful Statements

SELECT Functions

```
SELECT AVG(grade) FROM students;
SELECT SUM(grade) FROM students;
SELECT MIN(grade) FROM students;
SELECT MAX(grade) FROM students;
SELECT count(id) FROM students;
```

There are various functions that you can apply to the values returned from a select statement (many more on the reference pages).

```
sqlite> select COUNT(*) FROM Player;
COUNT(*)
125
sqlite> select SUM(age) FROM Player;
SUM(age)
3560
sqlite> select AVG(age) FROM Player;
AVG(age)
28.48
sqlite> select MIN(age) FROM Player;
MIN(age)
20
sqlite> select MAX(AGE) FROM Player;
```

SELECT Like

SELECT * FROM Player WHERE name LIKE "K%";

You can use like to do partial string matching. The '%;' character acts like a wild card.
The above will select all players whose name begins with K.

```
sqlite> SELECT * FROM Player WHERE name LIKE "K%";
name|position|number|team|age
Koji Uehara|1|19|Red Sox|40
Kevin Gausman|1|39|Orioles|24
Kevin Pillar|7|11|Blue Jays|26
Kevin Jepsen|1|40|Rays|30
```

ORDER BY

SELECT * FROM Player ORDER BY name;

Putting 'ORDER BY' at the end of a select statement will sort the output by the column name. There is also a 'REVERSE ORDER BY' which does the opposite sorted order (note that it looks like this is not working in sqlite3).

```
sqlite> SELECT * FROM Player ORDER BY name;
Aaron Loup|1|62|Blue Jays|27
Aaron Sanchez|1|41|Blue Jays|22
Adam Jones|8|10|Orioles|29
Adam Warren|1|43|Yankees|27
...
Wei-Yin Chen|1|16|Orioles|30
Xander Bogaerts|6|2|Red Sox|22
Xavier Cedeno|1|31|Rays|28
Zach Britton|1|53|Orioles|27
```

ORDER BY

SELECT * FROM Player ORDER BY team, name

You can order by multiple columns. The above will first sort the rows by team, and then within each team sort by name.

```
sqlite> SELECT * FROM Player ORDER BY team, name LIMIT 10;
Aaron Loup|1|62|Blue Jays|27
Aaron Sanchez|1|41|Blue Jays|22
Brett Cecil|1|27|Blue Jays|28
Dalton Pompey|8|45|Blue Jays|22
Daniel Norris|1|32|Blue Jays|21
Danny Valencia|5|23|Blue Jays|30
Devon Travis|4|29|Blue Jays|24
...
```

SELECT LIMITs

SELECT * FROM Player ORDER BY team, name LIMIT 10;

You can add a LIMIT qualifier at the end of a SELECT statement to only show the first X rows returned.

```
sqlite> SELECT * FROM Player ORDER BY team, name LIMIT 10;
Aaron Loup|1|62|Blue Jays|27
Aaron Sanchez|1|41|Blue Jays|22
Brett Cecil|1|27|Blue Jays|28
Dalton Pompey|8|45|Blue Jays|22
Daniel Norris|1|32|Blue Jays|21
Danny Valencia|5|23|Blue Jays|30
Devon Travis|4|29|Blue Jays|24
Drew Hutchison|1|36|Blue Jays|24
Edwin Encarnacion|3|10|Blue Jays|32
Jeff Francis|1|35|Blue Jays|34
```

SELECT LIMITs

SELECT * FROM Player ORDER BY team, name LIMIT 20, 10;

You can also add an offset to the LIMIT statement. The above skips the first 20 rows, then displays the following 10, i.e., it displays rows 20-29 (assuming the first is row 0).

```
sqlite> SELECT * FROM Player REVERSE ORDER BY team, name LIMIT 20, 10;
Miguel Castro|1|51|Blue Jays|20
R.A. Dickey|1|43|Blue Jays|40
Roberto Osuna|1|54|Blue Jays|20
Russell Martin|2|55|Blue Jays|32
Ryan Goins|4|17|Blue Jays|27
Adam Jones|8|10|Orioles|29
Alejandro De Aza|7|12|Orioles|30
Brad Brach|1|35|Orioles|28
Brian Matusz|1|17|Orioles|28
Bud Norris|1|25|Orioles|30
```

UPDATE

UPDATE Player SET age = 99 WHERE name = "Kevin Kiermaier";

UPDATE (along with where) allows you to update values in the table. If you don't have a WHERE clause, it will update all values to the given one.

```
sqlite> SELECT * FROM Player WHERE name = "Kevin Kiermaier";
name|position|number|team|age
Kevin Kiermaier|9|39|Rays|24

sqlite> UPDATE Player SET age = 99 WHERE name = "Kevin Kiermaier";

sqlite> SELECT * FROM Player WHERE name = "Kevin Kiermaier";
name|position|number|team|age
Kevin Kiermaier|9|39|Rays|99
```

UPDATE

UPDATE Player SET age = age - 10 WHERE name LIKE "K%";

You can also use some math (and other functions) within your UPDATE statements. The above reduces the selected players ages by 10.

```
sqlite> SELECT * FROM Player WHERE name like "K%";
name|position|number|team|age
Koji Uehara|1|19|Red Sox|40
Kevin Gausman|1|39|Orioles|24
Kevin Pillar|7|11|Blue Jays|26
Kevin Jepsen|1|40|Rays|30
sqlite>
sqlite> UPDATE Player SET age = age - 10 WHERE name LIKE "K%";
sqlite>
sqlite> SELECT * FROM Player WHERE name like "K%";
name|position|number|team|age
Koji Uehara|1|19|Red Sox|30
Kevin Gausman|1|39|Orioles|14
Kevin Pillar|7|11|Blue Jays|16
Kevin Jepsen|1|40|Rays|20
```

DELETE

DELETE FROM Player WHERE name = "Kevin Kiermaier";

DELETE allows you to delete rows from the table that satisfy the WHERE clause.

```
sqlite> SELECT * FROM Player WHERE name = "Kevin Kiermaier";
name|position|number|team|age
Kevin Kiermaier|9|39|Rays|99
sqlite>
sqlite> DELETE FROM Player WHERE name = "Kevin Kiermaier";
sqlite>
sqlite> SELECT * FROM Player WHERE name = "Kevin Kiermaier";
sqlite>
```

ALTER TABLE

ALTER TABLE Player ADD COLUMN
city text;

ALTER TABLE allows you to modify the columns within a table. It has a lot of options and the above is an example of adding a new column (called city of type text) to a table.

```
sqlite> ALTER TABLE Player ADD COLUMN
city text;
sqlite>
sqlite> select * from player limit 5;
name|position|number|team|age|city
Craig Breslow|1|32|Red Sox|34|
Clay Buchholz|1|11|Red Sox|30|
Joe Kelly|1|56|Red Sox|26|
Justin Masterson|1|63|Red Sox|30|
Wade Miley|1|20|Red Sox|28|
sqlite>
sqlite> UPDATE Player SET city =
"Rochester";
sqlite>
sqlite> select * from player limit 5;
name|position|number|team|age|city
Craig Breslow|1|32|Red Sox|34|Rochester
Clay Buchholz|1|11|Red Sox|30|Rochester
Joe Kelly|1|56|Red Sox|26|Rochester
Justin Masterson|1|63|Red Sox|30|Rochester
Wade Miley|1|20|Red Sox|28|Rochester
```

DROP TABLE

DROP TABLE Player;

DROP TABLE removes a table and all of its entries from the database. Be very careful when using it!

```
sqlite> DROP TABLE Player;
sqlite> SELECT * FROM Player;
Error: no such table: Player
```

Database Info

.tables

.tables
show tables;

In sqlite, the .tables command shows all tables in the database. In mysql, "show tables;" does the same.

```
sqlite> .read Baseball-AL-East-2015.sqlite
sqlite>
sqlite> .tables
Coach     Player    Position  Team
```

.schema

.schema Coach
show create table Coach;

In sqlite, the .schema command shows how the specified table was created. In mysql, you can use "show create table".

```
sqlite> .schema Coach
CREATE TABLE Coach (
  number INTEGER,
  name TEXT PRIMARY KEY,
  title TEXT,
  team TEXT REFERENCES Team(name)
);
sqlite>
sqlite> .schema Player
CREATE TABLE Player(
  name TEXT,
  position INTEGER REFERENCES Position(posnum),
  number INTEGER,
  team TEXT REFERENCES Team(name),
  age INTEGER,
  PRIMARY KEY(team, number)
);
```

Joins

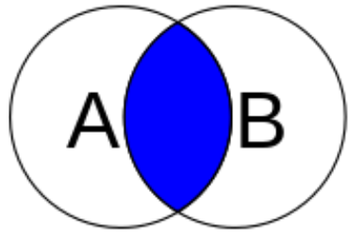
Joins

Joins allow the combination of fields from multiple tables. There are a number of ways to do this. We'll use the following two tables as an example (the .sql file is linked on the webpage).

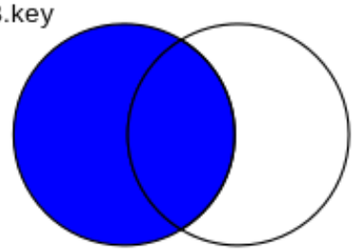
```
CREATE TABLE T1 (
    v1 text,
    v2 text,
    id int
);

CREATE TABLE T2 (
    v1 text,
    v2 text,
    id int
);

INSERT INTO T1 VALUES("A1", "B1", 1);
INSERT INTO T1 VALUES("C1", "D1", 2);
INSERT INTO T1 VALUES("E1", "F1", 3);
INSERT INTO T1 VALUES("G1", "H1", 4);

INSERT INTO T2 VALUES("A2", "B2", 3);
INSERT INTO T2 VALUES("C2", "D2", 4);
INSERT INTO T2 VALUES("E2", "F2", 5);
INSERT INTO T2 VALUES("G2", "H2", 6);
```

Joins

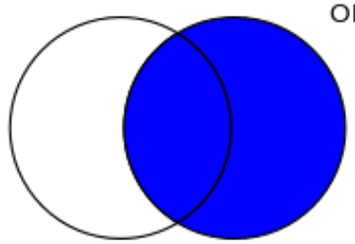
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key



SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key

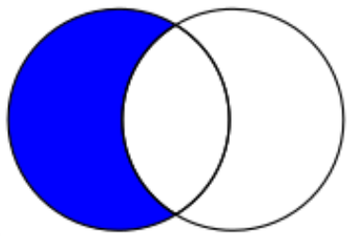


SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key

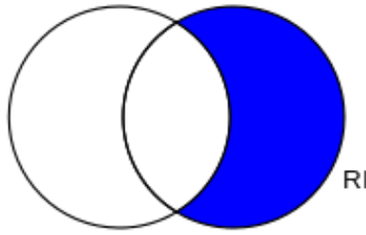


SQL JOINS

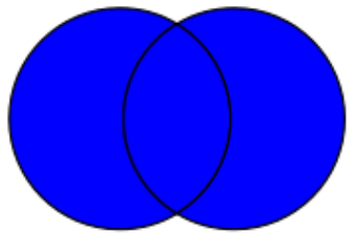
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL



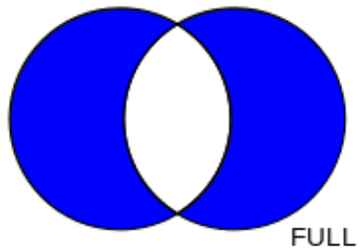
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL



SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key

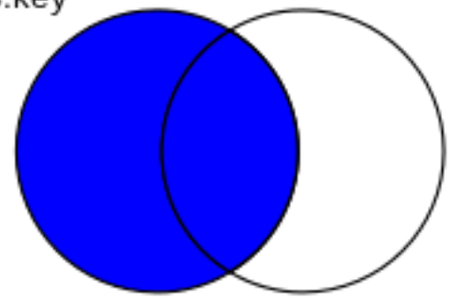


SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL

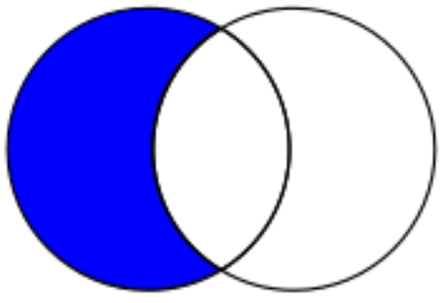


Left Joins

```
SELECT <fields>
FROM TableA  A
LEFT JOIN TableB  B
ON A.key = B.key
```



```
SELECT <fields>
FROM TableA  A
LEFT JOIN TableB  B
ON A.key = B.key
WHERE B.key IS NULL
```



Left Joins take the left table (i.e., the table before the JOIN statement) as the base and link it to the table on the right.

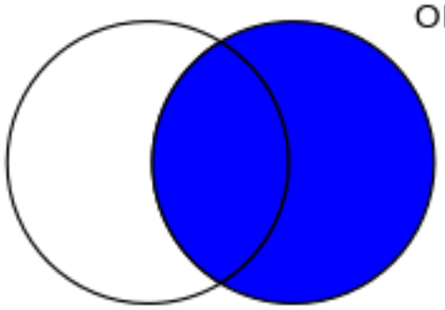
The first options selects everything in A, with everything that references with B; with nulls filled in for entries in A that have no reference to B.

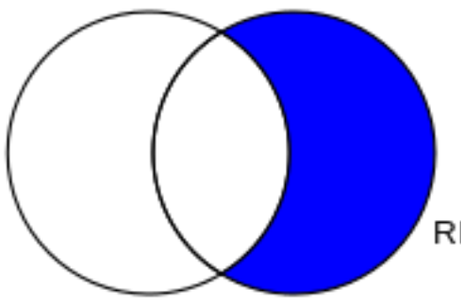
The second option selects everything in A that has no reference in B.

```
SELECT T1.v1, T1.v2, T2.v1, T2.v2 FROM T1 LEFT JOIN T2 ON T1.id = T2.id;
A1|B1||
C1|D1||
E1|F1|A2|B2
G1|H1|C2|D2

SELECT T1.v1, T1.v2, T2.v1, T2.v2 FROM T1 LEFT JOIN T2 ON T1.id = T2.id WHERE T2.id IS NULL;
A1|B1||
C1|D1||
```

Right Joins

```
SELECT <fields>
FROM TableA  A
RIGHT JOIN TableB  B
ON A.key = B.key
```





```
SELECT <fields>
FROM TableA  A
RIGHT JOIN TableB  B
ON A.key = B.key
WHERE A.key IS NULL
```

Right Joins take the Right table (i.e., the table after the JOIN statement) as the base and link it to the table on the left. They are the opposite of a left join.

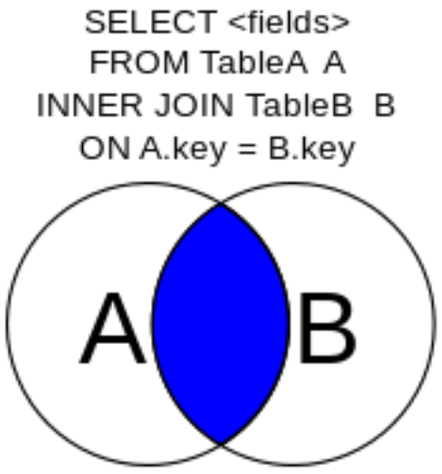
The first options selects everything in B, with everything that references with A; with nulls filled in for entries in B that have no reference to A.

The second option selects everything in B that has no reference in A.

Right joins are not yet implemented in sqlite, but you can just use a left join instead.

```
Error: near line 29: RIGHT and FULL OUTER JOINs are not currently supported

Error: near line 31: RIGHT and FULL OUTER JOINs are not currently supported
```

Inner Joins

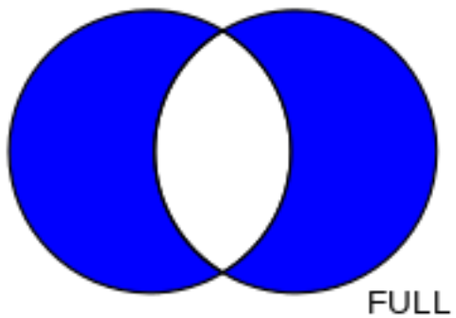


Inner joins are (at least in my experience) the most useful and most commonly used.

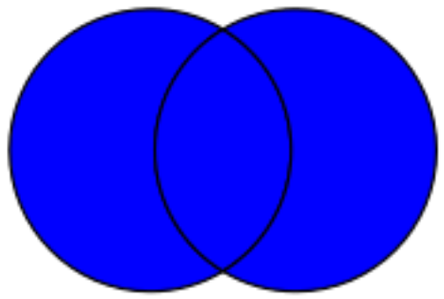
They provide the requested fields that exist in both A and B given the JOIN criteria.

```
SELECT T1.v1, T1.v2, T2.v1, T2.v2 FROM T1 INNER JOIN T2 ON T1.id = T2.id;
E1|F1|A2|B2
G1|H1|C2|D2
```

Outer Joins



```
SELECT <fields>
FROM TableA  A
FULL OUTER JOIN TableB  B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```



```
SELECT <fields>
FROM TableA  A
FULL OUTER JOIN TableB  B
ON A.key = B.key
```

Outer joins provide the last options. They can show either all the entries not referenced in both tables; or all the entries in both (linked by the reference when applicable).

These are very rarely used, and not implemented in many DBs. In part due to how they can create a massive table of results.

```
Error: near line 35: RIGHT and FULL OUTER JOINs are not currently supported

Error: near line 37: RIGHT and FULL OUTER JOINs are not currently supported
```