

## Additional SQL example

1. Create a table to store information about US weather observation stations, identified by an ID:

```
CREATE TABLE STATION
  (ID INTEGER PRIMARY KEY,
   CITY CHAR(20),
   STATE CHAR(2),
   LAT_N REAL,
   LONG_W REAL);
```

2. Add a few rows in the table STATION:

```
INSERT INTO STATION
  VALUES (13, 'Phoenix', 'AZ', 33, 112);
```

```
INSERT INTO STATION
  VALUES (44, 'Denver', 'CO', 40, 105);
```

```
INSERT INTO STATION
  VALUES (66, 'Caribou', 'ME', 47, 68);
```

3. Simple query to look at table STATION in undefined order:

```
SELECT * FROM STATION;
```

ID	CITY	STATE	LAT_N	LONG_W
13	Phoenix	AZ	33	112
44	Denver	CO	40	105
66	Caribou	ME	47	68

4. Query to select Northern stations (Northern latitude > 39.7): (restriction)

```
SELECT * FROM STATION
  WHERE LAT_N > 39.7;
```

ID	CITY	STATE	LAT_N	LONG_W
44	Denver	CO	40	105
66	Caribou	ME	47	68

5. Query to select only ID, CITY, and STATE columns: (projection)

```
SELECT ID, CITY, STATE FROM STATION;
```

ID	CITY	STATE
13	Phoenix	AZ
44	Denver	CO
66	Caribou	ME

6. Query combining the restriction and projection above:

```
SELECT ID, CITY, STATE FROM STATION  
WHERE LAT_N > 39.7;
```

ID	CITY	STATE
44	Denver	CO
66	Caribou	ME

7. Create another table, to store normalized temperature and precipitation data. Rows are identified by month and station ID. Temperatures are in degrees Fahrenheit and rainfalls in inches, and ensure realistic ranges:

```
CREATE TABLE STATS  
(ID INTEGER REFERENCES STATION(ID),  
MONTH INTEGER CHECK (MONTH BETWEEN 1 AND 12),  
TEMP_F REAL CHECK (TEMP_F BETWEEN -80 AND 150),  
RAIN_I REAL CHECK (RAIN_I BETWEEN 0 AND 100),  
PRIMARY KEY (ID, MONTH));
```

8. Populate the table STATS with some statistics for January and July:

```
INSERT INTO STATS  
VALUES (13, 1, 57.4, 0.31);
```

```
INSERT INTO STATS  
VALUES (13, 7, 91.7, 5.15);
```

```
INSERT INTO STATS  
VALUES (44, 1, 27.3, 0.18);
```

```
INSERT INTO STATS  
VALUES (44, 7, 74.8, 2.11);
```

```
INSERT INTO STATS
VALUES (66, 1, 6.7, 2.10);
```

```
INSERT INTO STATS
VALUES (66, 7, 65.8, 4.52);
```

9. Simple query to look at table STATS in undefined order:

```
SELECT * FROM STATS;
```

ID	MONTH	TEMP_F	RAIN_I
13	1	57.4	.31
13	7	91.7	5.15
44	1	27.3	.18
44	7	74.8	2.11
66	1	6.7	2.1
66	7	65.8	4.52

10. Query to look at table STATS, picking up location information by joining with table STATION on the ID column:

```
SELECT STATION.ID, STATION.CITY, STATION.STATE, STATS.MONTH,
STATS.TEMP_F, STATS.RAIN_I
FROM STATION, STATS
WHERE STATION.ID = STATS.ID;
```

ID	CITY	ST	MONTH	TEMP_F	RAIN_I
13	Phoenix	AZ	1	57.4	.31
13	Phoenix	AZ	7	91.7	5.15
44	Denver	CO	1	27.3	.18
44	Denver	CO	7	74.8	2.11
66	Caribou	ME	1	6.7	2.1
66	Caribou	ME	7	65.8	4.52

11. Query to look at the table STATS, ordered by month and greatest rainfall, (with columns rearranged):

```
SELECT MONTH, ID, RAIN_I, TEMP_F FROM STATS
      ORDER BY MONTH, RAIN_I DESC;
```

MONTH	ID	RAIN_I	TEMP_F
1	66	2.1	6.7
1	13	.31	57.4
1	44	.18	27.3
7	13	5.15	91.7
7	66	4.52	65.8
7	44	2.11	74.8

12. Query to look at temperatures for July from table STATS, lowest temperatures first, picking up city name and latitude by joining with table STATION:

```
SELECT LAT_N, CITY, TEMP_F
      FROM STATS, STATION
      WHERE MONTH = 7
            AND STATS.ID = STATION.ID
      ORDER BY TEMP_F;
```

LAT_N	CITY	TEMP_F
47	Caribou	65.8
40	Denver	74.8
33	Phoenix	91.7

13. Query to show MAX and MIN temperatures as well as average rainfall for each station:

```
SELECT MAX(TEMP_F), MIN(TEMP_F), AVG(RAIN_I), ID
      FROM STATS
      GROUP BY ID;
```

MAX(TEMP_F)	MIN(TEMP_F)	AVG(RAIN_I)	ID
91.7	57.4	2.73	13
74.8	27.3	1.145	44
65.8	6.7	3.31	66

14. Query to show stations with year-round average temperature above 50 deg.: (nested)

```
SELECT * FROM STATION
      WHERE 50 < (SELECT AVG(TEMP_F)
                  FROM STATS
                  WHERE STATION.ID = STATS.ID);
```

ID	CITY	ST	LAT_N	LONG_W
13	Phoenix	AZ	33	112
44	Denver	CO	40	105

15. Create a view to convert Fahrenheit to Celsius and inches to millimeters:

```
CREATE VIEW METRIC_STATS (ID, MONTH, TEMP_C, RAIN_M) AS
  SELECT ID,
         MONTH,
         (TEMP_F - 32) * 5 / 9,
         RAIN_I * 25.4
  FROM STATS;
```

16. Query to look at table STATS in a metric light (i.e. through the new view):

```
SELECT * FROM METRIC_STATS;
```

ID	MONTH	TEMP_C	RAIN_M
13	1	14.1111111	7.874
13	7	33.1666667	130.81
44	1	-2.6111111	4.572
44	7	23.7777778	53.594
66	1	-14.0555556	53.34
66	7	18.7777778	114.808

17. Another "metric query", restricted to January below-freezing (0 Celsius) data, sorted on rainfall:

```
SELECT * FROM METRIC_STATS
      WHERE TEMP_C < 0
            AND MONTH = 1
      ORDER BY RAIN_M;
```

ID	MONTH	TEMP_C	RAIN_M
44	1	-2.6111111	4.572
66	1	-14.0555556	53.34

18. Update all rows of table STATS to compensate for faulty rain gauges known to read 0.01 inches low: (and display the resulting table)

```
UPDATE STATS
  SET RAIN_I = RAIN_I + 0.01;
```

```
SELECT * FROM STATS;
```

ID	MONTH	TEMP_F	RAIN_I
13	1	57.4	.32
13	7	91.7	5.16
44	1	27.3	.19
44	7	74.8	2.12
66	1	6.7	2.11
66	7	65.8	4.53

19. Update one row, Denver's July temperature reading, to correct a data entry error: (and display the resulting table)

```
UPDATE STATS
  SET TEMP_F = 74.9
  WHERE ID = 44
     AND MONTH = 7;
```

```
SELECT * FROM STATS;
```

ID	MONTH	TEMP_F	RAIN_I
13	1	57.4	.32
13	7	91.7	5.16
44	1	27.3	.19
44	7	74.9	2.12
66	1	6.7	2.11
66	7	65.8	4.53

20. Delete July data and East Coast data from both tables: (and display the resulting tables)

```
DELETE FROM STATS
  WHERE MONTH = 7
  OR ID IN (SELECT ID FROM STATION
            WHERE LONG_W < 90);
```

```
DELETE FROM STATION
  WHERE LONG_W < 90;
```

```
SELECT * FROM STATION;
```

ID	CITY	STATE	LAT_N	LONG_W
13	Phoenix	AZ	33	112
44	Denver	CO	40	105

```
SELECT * FROM STATS;
```

ID	MONTH	TEMP_F	RAIN_I
13	1	57.4	.32
44	1	27.3	.19

21. Attempt to insert a row for an unknown observation station: (ID value of 33 does not match a station ID value in the STATION table. This is a violation of referential integrity)

```
INSERT INTO STATS
  VALUES (33,8,27.4,.19);
```

error message

violation of constraint STATS\_FOREIGN1 caused operation to fail

22. Attempt to update a row with a temperature below the range -80 TO 150:

```
UPDATE STATS
  SET TEMP_F = -100
  WHERE ID = 44
  AND MONTH = 1;
```

error message

violation of constraint STATS\_CHECK2 caused operation to fail

23. Attempt to insert a row with negative rainfall measurement, outside the range 0 to 100:

```
INSERT INTO STATS  
VALUES (44,8,27.4,-.03);
```

error message

violation of constraint STATS\_CHECK3 caused operation to fail

24. Attempt to insert a row with month 13, outside the range of 1 to 12:

```
INSERT INTO STATS  
VALUES (44,13,27.4,.19);
```

error message

violation of constraint STATS\_CHECK1 caused operation to fail

25. Attempt to insert a row with a temperature above the range -80 TO 150:

```
INSERT INTO STATS  
VALUES (44,8,160,.19);
```

error message

violation of constraint STATS\_CHECK2 caused operation to fail

26. Attempt to insert a row with no constraint violations: (and display the resulting table)

```
INSERT INTO STATS  
VALUES (44,8,27.4,.10);
```

status message

1 row inserted

```
SELECT * FROM STATS;
```

ID	MONTH	TEMP_F	RAIN_I
44	8	27.4	.10
13	1	57.4	.32
44	1	27.3	.19

27. Attempt to insert a second row of August statistics for station 44: (violation of the primary key constraint)

```
INSERT INTO STATS  
VALUES (44,8,160,.19);
```

error message

violation of constraint STATS\_PRIMARY\_ID\_MONTH caused operation to fail