

Part 1 – select, from, where, order by; and, or, <>m is, is not, distinct, length, *, like, %, ‘’

Part 2 – join/on, implicit join, alias, inner (results: overlapped) vs left/outer vs right/outer join vs full/outer (result for outer joins: null values could be there), cross join, group by

Part 3 – data types, math, other features like LOWER(), UPPER(), SUBSTR(), as

Part 4 – insert into, values; update, set, where; delete from

Part 1 – select, from, where, order by; and, or, <>m is, is not, distinct, length, *, like, %, ‘’

```
SELECT * FROM people;
```

```
SELECT team, first_name, last_name FROM people WHERE state_code='CA' AND shirt_or_hat='shirt' AND team IS NOT 'Angry Ants';
```

```
SELECT team, first_name, last_name FROM people WHERE state_code='CA' AND shirt_or_hat='shirt' AND team <> 'Angry Ants';
```

```
SELECT team, first_name, last_name FROM people WHERE state_code='CA' OR state_code='CO' AND shirt_or_hat='shirt' AND team IS 'Angry Ants';
```

```
SELECT * FROM people WHERE company LIKE '%LLC' LIMIT 10 OFFSET 5;
```

```
SELECT first_name, LENGTH(first_name) FROM people;
```

```
SELECT DISTINCT(first_name) FROM people ORDER BY first_name;
```

```
SELECT COUNT(*) FROM people WHERE state_code='CA';
```

```
SELECT COUNT(first_name) FROM people WHERE state_code='CA';
```

Let's create a report that shows which members of each team in the state of Colorado (CO) want a shirt or a hat for participating in our quiz. We'll sort the results in a way that keeps the data organized: first, we'll sort the participants by team, in alphabetical order, and then we'll sort by whether they want a shirt or hat. This also will be an alphabetical sort. Within each of these team-apparel combinations, we'll sort the participants by their last name, in reverse alphabetical order (Z-A).
<further details/requirements not shown>

```
SELECT team, shirt_or_hat, first_name, last_name  
FROM people  
WHERE state_code = 'CO' AND (shirt_or_hat = 'shirt' OR shirt_or_hat = 'hat')  
ORDER BY team, shirt_or_hat, last_name DESC;
```

WHERE clause that returns all records with the text "priority" in the Comment column:

- WHERE Comment LIKE '%priority%'

The "Items" table has one "Name" column with 12 items in it. What will this query return?

```
SELECT 'Name' FROM Items;
```

- the text "Name" showing 12 times

note: Since "Name" is in quotes, it will be the actual text returned for each one of the 12 items.

Part 2 – join/on, implicit join, alias, inner (results: overlapped) vs left/outer vs right/outer join vs full/outer (result for outer joins: null values could be there), cross join, group by

Inner join to connect tables (selecting all fields from all tables here):

```
SELECT * FROM people JOIN states ON people.state_code=states.state_abbrev;
```

WRONG (each person/row is expanded to 50 rows [50 states]): `SELECT * FROM people JOIN states;`

A **cross join** produces a Cartesian product, combining every row from the first table with every row from the second table, and requires no matching condition.

Implicit join (explicit join above is more preferable though):

```
SELECT people.first_name, states.state_name  
FROM people, states  
WHERE people.state_code=states.state_abbrev;
```

Implicit join (alias used here):

```
SELECT ppl.first_name, st.state_name  
FROM people ppl, states st  
WHERE ppl.state_code=st.state_abbrev;
```

Left/outer join (NULL will appear if the state in people is NOT in the states table):

```
SELECT *  
FROM people  
LEFT JOIN states  
ON people.state=states.abbr;
```

people

name	state
Marcus	CA
Jennifer	VA
Devin	MA
Britt	CA

states

abbr	full
CA	California
DE	Delaware
VA	Virginia

result

name	state	abbr	full
Marcus	CA	CA	California
Jennifer	VA	VA	Virginia
Devin	MA	NULL	NULL
Britt	CA	CA	California

.. vs right/outer join:

people

name	state
Marcus	CA
Jennifer	VA
Devin	MA
Britt	CA

states

abbr	full
CA	California
DE	Delaware
VA	Virginia

result

name	state	abbr	full
Marcus	CA	CA	California
Jennifer	VA	VA	Virginia
Britt	CA	CA	California
NULL	NULL	DE	Delaware

```
SELECT * FROM people RIGHT JOIN states
ON people.state = states.abbr;
```

.. vs full/outer join (showing BOTH matched and unmatched records):

people

name	state
Marcus	CA
Jennifer	VA
Devin	MA
Britt	CA

states

abbr	full
CA	California
DE	Delaware
VA	Virginia

result

name	state	abbr	full
Marcus	CA	CA	California
Jennifer	VA	VA	Virginia
Devin	MA	NULL	NULL
Britt	CA	CA	California
NULL	NULL	DE	Delaware

```
SELECT * FROM people FULL OUTER JOIN states
ON people.state = states.abbr;
```

Listing states without people for them first (as those got NULL for people.state_code):

```
SELECT DISTINCT(people.state_code), states.state_name
FROM states
LEFT JOIN people
ON people.state_code=states.state_abbrev
ORDER BY people.state_code;
```

WRONG (this will result in 1 row):

```
SELECT first_name, COUNT(first_name) FROM people;
```

Correct (showing the count for each distinct first_name):

```
SELECT first_name, COUNT(first_name)
FROM people
GROUP BY first_name;
```

WRONG:

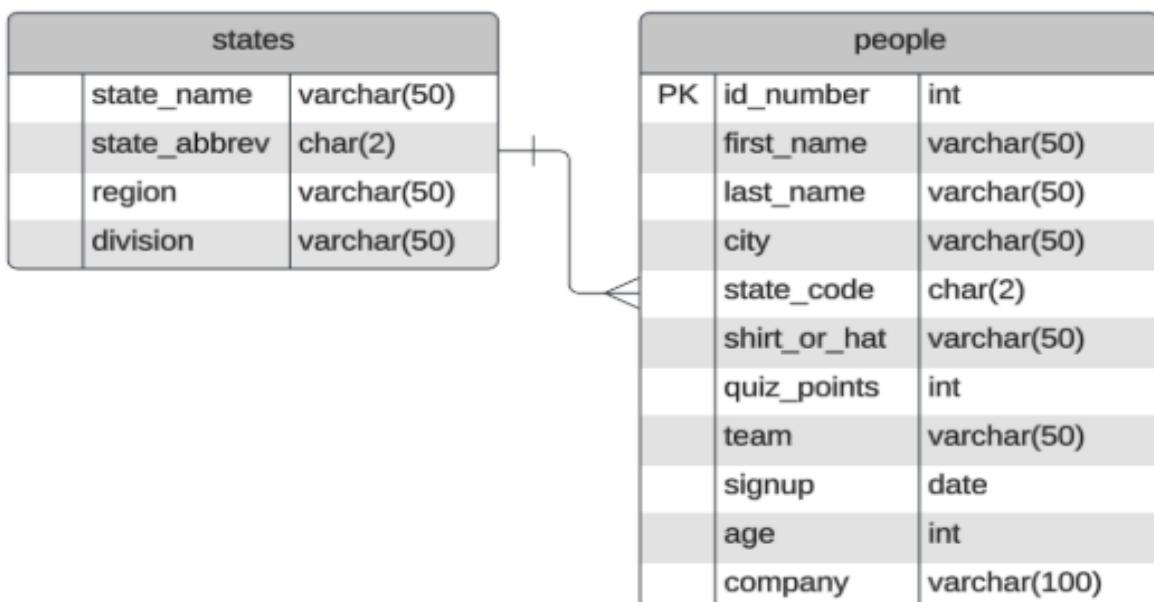
```
SELECT state_code, quiz_points, COUNT(quiz_points)
FROM people
GROUP BY quiz_points
```

Correct:

```
SELECT state_code, quiz_points, COUNT(quiz_points)
FROM people
GROUP BY state_code, quiz_points
```

Let's prepare a report showing how many members of each team live in each region of the country.

Our quiz database contains two tables, `states` and `people`:



```
SELECT states.region, people.team, COUNT(people.team)
FROM people
LEFT JOIN states
```

WRONG: `GROUP BY states.region, people.team;`

```

SELECT states.region, people.team, COUNT(people.team)
FROM people
LEFT JOIN states ON people.state_code=states.state_abbrev
GROUP BY states.region, people.team;

```

Correct:

Part 3 – data types, math, other features like LOWER(), UPPER(), SUBSTR(), as

Binary	Date/Time	Numbers	Text
BINARY	DATE	BIGINT	CHARACTER
BINARY LARGE OBJECT	INTERVAL	DECIMAL	CHARACTER LARGE OBJECT
BINARY VARYING	TIME	DOUBLE PRECISION	VARCHAR
	TIMESTAMP	FLOAT	NCHAR
		INTEGER	NCHAR VARYING
		NUMERIC	
Other		SMALLINT	
BOOLEAN		REAL	

Null - a field having no value; NOT the same as false, no, or 0

+,-,*,/,%

Assumes int ops unless otherwise specified

Comparison ops: >,<,>=,<=,=,!/,<>

SUM(), AVG(), etc

SELECT 10+1;

SELECT 1/3;

SELECT 1/3.0;

```

SELECT MAX(quiz_points), MIN(quiz_points)
FROM people;

```

NO:

```

SELECT team, COUNT(*), SUM(quiz_points), SUM(quiz_points)/COUNT(*)

```

FROM people

GROUP BY team;

YES:

```

SELECT team, COUNT(*), SUM(quiz_points), AVG(quiz_points)

```

FROM people

GROUP BY team;

WRONG:

```

SELECT first_name, last_name, quiz_points FROM people WHERE quiz_points=MAX(quiz_points);

```

Correct:

```

SELECT first_name, last_name, quiz_points FROM people WHERE quiz_points=(SELECT MAX(quiz_points)
FROM people);

```

Assuming you don't remember the exact state code for Minnesota:

```
SELECT *
FROM people
WHERE state_code=(  
SELECT state_abbrev FROM states WHERE state_name='Minnesota'  
);
```

Substring – 1st 5 chars:

```
SELECT LOWER(first_name), SUBSTR(last_name, 1, 5) FROM people;
```

Substring – from 1st char all the way to the end:

```
SELECT LOWER(first_name), SUBSTR(last_name, 1) FROM people;
```

Substring – last 2 chars:

```
SELECT LOWER(first_name), SUBSTR(last_name, 1, -2) FROM people;
```

Replace is case-sensitive:

```
SELECT REPLACE(first_name, 'a', '-') FROM people;
```

This will NOT show the highest score of 100 as 99 is the “max” in char:

```
SELECT MAX(CAST(quiz_points AS CHAR)) FROM people;
```

This will be correctly show 100 as the highest score:

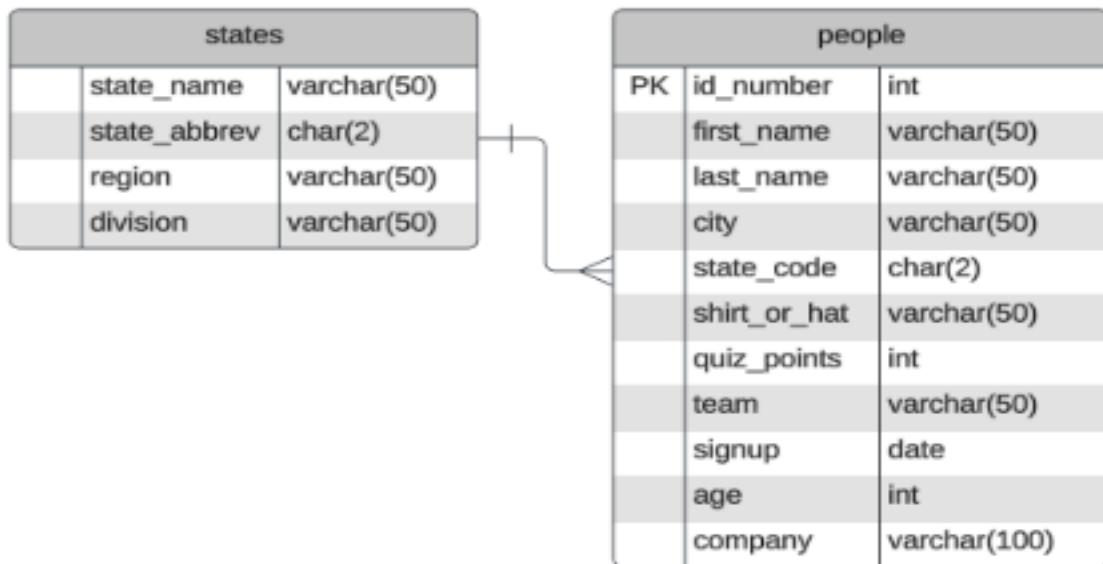
```
SELECT MAX(CAST(quiz_points AS INT)) FROM people;
```

Alias -> as:

```
SELECT first_name as firstname, UPPER(last_name) as surname FROM people;
```

Let's create a report showing the average score and maximum score earned by participants in each state.

Our quiz database contains two tables, `states` and `people`:



Your task: Write a query that returns the maximum and average scores for each state, along with the state's name, sorted from highest to lowest average score.

Result

Your result should have three columns:

- `STATE`: the name of each state
- `MAXPOINTS`: the highest value of `quiz_points` for each state
- `AVGPOINTS`: the average number of `quiz_points` earned by participants in each state

Sort the results by `AVGPOINTS` from highest to lowest.

WRONG:

```
SELECT states.STATE_NAME, MAX(people.QUIZ_POINTS), AVG(people.QUIZ_POINTS)
FROM people
JOIN states ON people.STATE_CODE=states.STATE_ABBREV
ORDER BY AVG(people.QUIZ_POINTS) DESC
GROUP BY people.STATE_CODE;
```

Correct(last 2 rows swapped, and with alias):

```
SELECT states.STATE_NAME, MAX(people.QUIZ_POINTS), AVG(people.QUIZ_POINTS) as avgpts
FROM people JOIN states ON people.STATE_CODE=states.STATE_ABBREV
GROUP BY people.STATE_CODE
ORDER BY avgpts DESC;
```

Part 4 – insert into, values; update, set, where; delete from

```
INSERT INTO people (first_name) VALUES ('Bob');
```

```
INSERT INTO people  
(first_name, last_name, state_code, city, shirt_or_hat)  
VALUES  
('Mary', 'Hamilton', 'OR', 'Portland', 'hat');
```

WRONG:

```
INSERT INTO people  
(first_name, last_name)  
VALUES  
('George', 'White'),  
('Jenn', 'Smith'),  
('Carol');
```

Correct:

```
INSERT INTO people  
(first_name, last_name)  
VALUES  
('George', 'White'),  
('Jenn', 'Smith'),  
('Carol', NULL);
```

```
UPDATE people SET last_name = 'Morrison' WHERE last_name='Morrison';
```

```
SELECT last_name FROM people WHERE first_name='Carlos' AND city='Houston';
```

```
UPDATE people SET last_name='Morrison' WHERE first_name='Carlos' AND city='Houston';
```

```
SELECT * FROM people WHERE id_number=175;
```

```
UPDATE people SET last_name='Morrison' WHERE id_number=175;
```

```
DELETE FROM people WHERE id_number=1001;
```

```
DELETE FROM people WHERE quiz_points IS NULL;
```

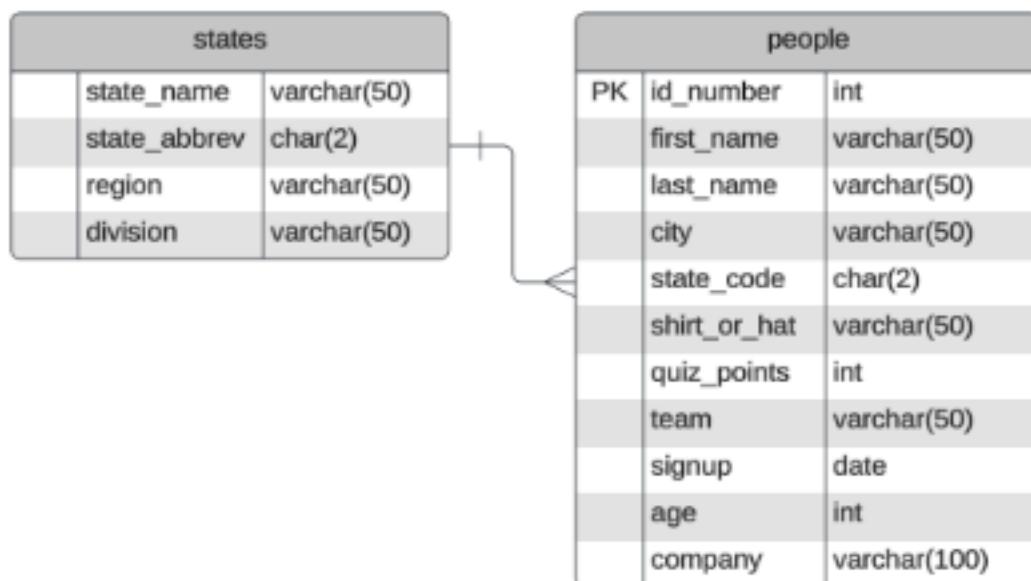
Identifying the correct record

Before we modify or delete an existing record, we should check that we're using the correct one, and that our selection logic doesn't affect other rows we don't intend to change.

We'll usually want to refer to rows by unique identifiers, rather than field values like names or other things that might be shared by two or more records.

To find a record's unique identifier, we can write a SELECT statement with information we know that allows us to narrow down results and refer to only the record we want to work with.

Our quiz database has two tables, `states` and `people`:



Your task: Create a SQL statement that shows the `id_number` and other confirmatory information for Alice, from Florida, who is a member of the Cobras.
<other req'ts not shown> .. since “Cobras” is NOT the team’s full name ..

```
SELECT people.ID_NUMBER, people.FIRST_NAME, people.STATE_CODE, people.TEAM  
FROM people JOIN states ON people.STATE_CODE=states.STATE_ABBREV  
WHERE people.FIRST_NAME='Alice' AND states.state_name='Florida' AND people.team like '%Cobras%';
```

- Text values should be in single quotation marks (')
- Field names with spaces need to be in backticks (`),
but avoid spaces in field names if possible
- Smart quotes (‘), rather than straight quotes ('),
will cause problems
- Share SQL statements as plain-text .sql files or attachments
rather than in chat or email

Use: IS NULL .. or IS NOT NULL .. i.e. don't use =