DDL: Commands act upon the schema.

```
CREATE TABLE <table name> (<attribute name> <type> <constraint list>)
DROP TABLE <table name>
ALTER TABLE <table name> ADD <attribute name> <type> <constraint list>
ALTER TABLE <table name> DROP <attribute name>
```

DML: Commands act upon the data.

```
UPDATE colleges SET abbr = 'COSAM' WHERE abbr = 'COASM'
DELETE FROM <table list> WHERE <condition> -- just table name deletes all rows
INSERT INTO <table list> VALUES <value list>
```

Relational Algebra

Doing these removes duplicates since these are sets. Try not to look through entire table when solving problems.

- Selection - $\sigma_{\mathrm{C}}(R)$ - returns rows that satisfy C
- Projection - $\pi_{\text{attributes}}(R)$ - returns columns that are in attribute list, no duplicates
- Cartesian Product - $R \times S$ - returns all combinations of rows (match every row in R with every row in S)
- Rename - $\rho_{\text{new}}(R)$ - used to do self joins, once original renamed they are forgotten for the duration of the operation
- Duplication Elimination - $\delta(R)$ - removes duplicates, enables us to between set operations and bag operations
- Sort - $\tau_{\mathrm{C}}(R)$ - sorts rows of R based on C, $F = Desc(A\%B), B, Desc(C)$
- Group By and Aggregate - $\gamma_{\mathrm{L}}(R)$ - groups rows of R, where L is a list of attributes, and applies aggregate functions to each group, **Agg. func. show up in other clauses should still be in L b/c projection would remove from final result**
- Outer Joins - $R \rtimes\!\bowtie S$ - returns all rows of R and matching rows of S, **includes tuples with no match**, same for right outer join, full outer join is both left and right outer join

Joins

- Theta Join (Equi-Join) - $R \bowtie_{\mathrm{C}} S$ - returns all combinations of rows that satisfy C, compare every combination **Keep columns since there is no projection**, $R \bowtie_{\Theta} S = \sigma_{\Theta}(R \times S)$
- Natural Join - $R \bowtie S$ - returns all combinations of rows that match on common attributes, **removes one set of common attributes from the final relation**
- Left/Right Semi Join - $R \ltimes \rtimes S$ - only attributes of one relation are kept, projection on all elements of one relation

Set Operations

Only apply these when R and S have the same schema. These are bag operations.

- Union - $R \cup S$ - combine rows of R and S, **remove duplicates**
- Set Difference - $R - S$ - keep rows that are unique to R
- Intersection - $R \cap S$ - keep rows that are in both R and S

SQL SELECT

```
6 - SELECT, 1 - FROM, 2 - WHERE, 3 - GROUP BY, 4 - HAVING, 5 - ORDER BY
```

$$\sigma_c(R) \rightarrow \text{SELECT * FROM R WHERE condition}$$
$$\pi_L(R) \rightarrow \text{SELECT L FROM R}$$
$$\pi_L(R) \rightarrow \text{SELECT DISTINCT L F FROM R}$$
$$\tau_L(R) \rightarrow \text{SELECT * FROM R ORDER BY L}$$
$$R \times S \rightarrow \text{SELECT * FROM R, S}$$
$$R \bowtie_C S \rightarrow \text{SELECT * FROM R, S WHERE condition}$$

```
SELECT Name FROM Pokemon WHERE Name LIKE 'B%' ORDER BY Name
```

• "LIKE" is a string operator. % is a wildcard. _ is a single character wildcard. Wildcard means any number of characters. **LIKE is case sensitive**.
• Can use "AS" to rename columns. Can also use in "ORDER BY" to sort by that renamed attribute. It **cannot be used** in "WHERE" because it is not a real attribute.

Group By and Aggregate

• If using Group By in SQL, then SELECT clause must contain only: Attributes that are listed in the GROUP BY clause, Aggregate operations on attributes not listed in the GROUP BY clause, COUNT(*)
• Cannot use aggregate functions in WHERE clause. Must use HAVING clause.

SQL Joins

```
SELECT * FROM Pokemon p JOIN Attributes a ON p.PokedexId = a.PokedexId
-OR- SELECT * FROM Pokemon p, Attributes a USING (PokedexId) (if joining on same attribute)
```

NATURAL JOIN will join on all attributes with the same name:

```
SELECT ... FROM <table1> NATURAL JOIN <table2>
```

OUTER JOIN:

```
SELECT ... FROM <table1> LEFT/RIGHT [OUTER] JOIN <table2> ON <condition>
```

Examples

IN:

```
SELECT Pokemon.Name, weight FROM Attributes, Pokemon WHERE Attributes.PokedexID IN
    (SELECT PokedexId FROM Species WHERE typeID = (SELECT typeId FROM Types WHERE
    type = 'ground')) AND Pokemon.PokedexId = Attributes.PokedexId
```

EXISTS:

```
SELECT DISTINCT type FROM Stats a WHERE EXISTS (SELECT * FROM Stats b WHERE a.type =
b.type AND weight > 3000)
```

GROUP_CONCAT:

```
SELECT grade, GROUP_CONCAT(DISTINCT classroom ORDER BY classroom ASC SEPARATOR ', ')
FROM list GROUP BY grade;
```

WITH:

```
WITH studentsClassroom AS (SELECT t.classroom, COUNT(l.FirstName) AS numStudents
    FROM teachers t, list l WHERE t.classroom = l.classroom GROUP BY t.classroom)
SELECT s1.classroom AS c1, s2.classroom AS c2, s1.numStudents
FROM studentsClassroom s1, studentsClassroom s2
WHERE s1.classroom < s2.classroom AND s1.numStudents = s2.numStudents
ORDER BY s1.numStudents;
```

Less than sign is used to avoid duplicate pairs.