

1 Overview

```
SELECT <attributes>
  FROM <tablenames>
 WHERE <condition>;
```

- <attributes> - the attributes to select
- <tablenames> - the table names to select from
- <condition> is a boolean expression on the attributes of the table

2 WHERE Condition

- <> \equiv not equals
- other operators include <, >, <=, >=, BETWEEN. (BETWEEN is inclusive.)

```
... WHERE max_credits BETWEEN 3 AND 6;
```

- compound expressions: AND, OR, NOT
- Testing for NULL: must use IS NULL or IS NOT NULL
- LIKE and NOT LIKE

```
... WHERE instructor LIKE 'Paint%';
```

- IN

```
... WHERE x IN (1, 2, 3);
```

3 Select Statements

3.1 Selecting Expressions on Attributes

```
SELECT 42 / 13 + 12; -- selects 15 (integer math)
SELECT a || ' ' || b || ' ' || c FROM foo; -- string concatenation
SELECT substring(a FROM 1 FOR 4) FROM foo; -- first four characters
```

3.2 Names and Aliasing

AS - used for renaming

```
SELECT substring(foo FROM 1 FOR 4) as f, bar as b FROM baz;
```

3.3 Schemas

```
-- given "project1" in cpainter and "project1" in public
SELECT * FROM public.project1; -- selects the public one
```

3.4 Misc

```
SELECT count(*) FROM mines_courses_meetings;
SELECT DISTINCT a1, a2, a3 ...
```

4 Joins

```
SELECT * FROM A, B WHERE A.x = B.x;
SELECT * FROM A JOIN B ON B.x = A.x; -- using join syntax
```

5 Order By

```
... ORDER BY attr DESC/ASC
```

6 Table Creation

```
CREATE TABLE [schema_name.]table_name
(
  attribute1 type1 NOT NULL, -- you can add constraints
  attribute2 type2 PRIMARY KEY,
  attribute3 type3
)
```

```
CREATE TABLE yourid.stuff (
  id serial PRIMARY KEY,
  stuff_id integer REFERENCES yourid.stuff(id), --foreign key inline
  name text NOT NULL,
  PRIMARY KEY (name, age) --implies not null constraints on name, age,

  -- Or can use this to declare foreign key
  FOREIGN KEY (stuff_id) REFERENCES yourid.stuff(id)
);
```

7 Types

- Integer Types
 - INTEGER (32 bit)
 - SMALLINT (16 bit)
 - BIGINT (64 bit)
 - SERIAL auto-incrementing integer

- Fixed precision numeric
 - NUMERIC(w, p)
 - DECIMAL(w, p) (an alias for NUMERIC)

Where w = width and p = precision. No more than w digits total, p after decimal point.

- Floating point
 - REAL - 32-bit
 - DOUBLE PRECISION - 64-bit

- Strings
 - CHAR(n) - strings of exactly n characters.
 - VARCHAR(n) - strings up to n characters (space padding not necessary)
 - (Oracle) VARCHAR2(n) - exactly like VARCHAR
 - (Postgres) TEXT - USE THIS - essentially infinite width and indexable

- Date/Time
 - DATE - holds dates. Standard format: 'YYYY-MM-DD'.
 - TIME - holds times. Standard format: 'HH:MM:SS' or 'HH:MM:SS.nnn' (fractional seconds). By default, this is time without timezone.
 - TIME WITH TIMEZONE - same as time, but with timezone
 - TIMESTAMP - time and date

- Typecasting

```
SELECT CAST('1/2/2016' AS DATE) AS foo;
```

8 INSERT, DELETE, UPDATE

```
INSERT INTO table VALUES (v1, v2, ...);
INSERT INTO table (a1, a2, ..., an) VALUES (v1, v2, ..., vn);
INSERT INTO table (a1, ..., an) VALUES (v1, ..., vn), (v1, ..., vn)
```

-- important to use WHERE condition here

```
DELETE FROM table WHERE condition;
```

```
UPDATE table_name SET a1 = v1, a2 = v2, ... WHERE condition;
```

9 Aggregate Functions

summarize data in a table for some column

- COUNT - counts the non-null entries in a column, or tuples

```
SELECT COUNT(*) FROM ...
```

Can be used with DISTINCT as well.

- SUM - adds up the entries in a column
- MAX/MIN - the maximum/minimum entry in a column
- AVG and others

9.1 GROUP BY

```
SELECT a1, a2, ..., f(a3), f(a4), ...  
FROM table1, table2, ... WHERE ...  
GROUP BY a1, a2, ...  
ORDER BY ...
```

a1, a2, ... must be in the GROUP BY.

10 Set Operations in SQL

10.1 Operators

- UNION - union of two sets of tuples (types and number of attributes must be compatible) - all rows in both relations
- INTERSECTION - all rows in common
- EXCEPT - set difference: all rows in the first relation **not in** the second relation

```
SELECT ... UNION SELECT ...;  
SELECT ... INTERSECTION SELECT ...;  
SELECT ... EXCEPT SELECT ...;
```

11 Examples

```
SELECT mc.instructor, mc.course_id,  
       mef.office, mef.email  
FROM mines_courses as mc, mines_eecs.faculty AS mec  
WHERE mc.instructor = mef.name;
```

```
SELECT * FROM foo WHERE bar = 3 ORDER BY alpha, beta, gamma;
```

11.1 Grouping

```
SELECT a1, a2, ..., f(a3), f(a4), ...  
FROM ... WHERE ...  
GROUP BY a1, a2, ...;
```

Result: divides tuples in relation into groups characterized by unique values of the group by attributes; aggregates are computed over the groups.

```
SELECT COUNT(*), instructor FROM mines_courses  
GROUP BY instructor  
ORDER BY COUNT(*) DESC;
```
