
# SQL programming overview

sol genomics network

# SQL (Structured Query Language)

Language for building, accessing, and manipulating a relational database management systems (RDBMS)

- Schema
- Tables
- Relationships

# SQL server basics

- Keywords are not case sensitive
- Table and column names are stored as they are entered (use lower case as naming convention)
- Elements are comma separated
- Comments are enclosed between /* and */ or preceded by -- .
- SQL statements are terminated by semicolon.

# SQL (Structured Query Language)

- Data definition
- Data manipulation
- SELECT statements
- Joins
- Row functions
- Aggregate functions
- Subqueries
- Views
- PostgreSQL, and database connection with Perl
- Resources

sol genomics network

# Data definition

## Data types

| Name | Aliases | Description |
|---|---|---|
| bigint | int8 | signed eight-byte integer |
| bigserial | serial8 | autoincrementing eight-byte integer |
| bit [ (*n*) ] | | fixed-length bit string |
| bit varying [ (*n*) ] | varbit | variable-length bit string |
| boolean | bool | logical Boolean (true/false) |
| box | | rectangular box in the plane |
| bytea | | binary data ("byte array") |
| character varying [ (*n*) ] | varchar [ (*n*) ] | variable-length character string |
| character [ (*n*) ] | char [ (*n*) ] | fixed-length character string |
| cidr | | IPv4 or IPv6 network address |
| circle | | circle in the plane |
| date | | calendar date (year, month, day) |
| double precision | float8 | double precision floating-point number |

http://www.postgresql.org/docs/8.3/static/datatype.html

sol genomics network

# Data definition

## Create table

Defines the structure of the table:

```
CREATE TABLE public.pubstatus (
 pubstatus_id serial NOT NULL PRIMARY KEY,
 pub_id integer NOT NULL REFERENCES public.pub,
 status boolean DEFAULT 1,
 date timestamp with time zone default now()
);
```

# Data definition

## Alter table

modifies the structure of the table:

```
ALTER TABLE allele ALTER COLUMN allele_symbol SET NOT NULL;

ALTER TABLE allele DROP COLUMN allele_gi;

ALTER TABLE locus ADD locus_description text DEFAULT NULL;
```

## Drop tables

```
DROP TABLE allele;
```

# Data definition

**Constraints are used to enforce valid data in columns**

modifies the structure of the table:

NOT NULL

CHECK

PRIMARY KEY

FOREIGN KEY

http://www.postgresql.org/docs/8.3/static/ddl-constraints.html

sol genomics network

# Data manipulation

## Insert – add new rows to your table

```
INSERT INTO sgn.accession
(organism_id, common_name, cxgn_production_visible)
VALUES (1,'M82', 'y');
```

## Update – modify column value/s of existing rows

```
UPDATE locus_alias SET obsolete ='f' WHERE locus_id = 105;
```

## Delete -remove rows from table

```
DELETE FROM locus WHERE locus_id = 99;
```

# Data manipulation

**Transactions**

BEGIN;

UPDATE.....;

DELETE.....;

INSERT .....;

COMMIT; or ROLLBACK

UPDATE locus_alias SET obsolete ='f';

# Data manipulation - transactions

```
cxgn=# begin;
BEGIN
cxgn=# UPDATE locus_alias SET obsolete='f';
UPDATE 6808
cxgn=#
cxgn=# rollback;
ROLLBACK
```

```
cxgn=# begin;
BEGIN
cxgn=# UPDATE locus_alias SET obsolete='f' WHERE locus_id=570;
UPDATE 1
cxgn=# commit;
COMMIT
```

# SELECT statements

- The SELECT statement retrieves data from database tables

SELECT *select_list (describes the columns of the result set.)*

[ INTO *new_table_name* ] (specifies that the result set is used to create a new table)

FROM *table_list (Contains a list of the tables from which the result set data is retrieved)*

[ WHERE *search_conditions* ] (filter that defines the conditions each row in the source tables must meet)

[ GROUP BY *group_by_list* ] (partitions the result set into groups based on the values)

[ HAVING *search_conditions* ] (an additional filter that is applied to the result set)

[ ORDER BY *order_list* [ ASC | DESC ] ] (defines the order in which the rows in the result set are sorted)

# SELECT statements

```
SELECT * FROM phenome.locus;

SELECT locus_id, locus_name FROM locus
WHERE common_name_id=1
ORDER BY locus_symbol ASC;
```

```
cxgn=# SELECT count(locus_id) , common_name FROM locus
JOIN sgn.common_name USING(common_name_id)
GROUP BY common_name
HAVING common_name in ('Tomato' , 'Potato' , 'Eggplant') ;
 count | common_name
-------+-------------
  1038 | Potato
  2216 | Tomato
    48 | Eggplant
(3 rows)
```

# SELECT statements

Conditional operators:

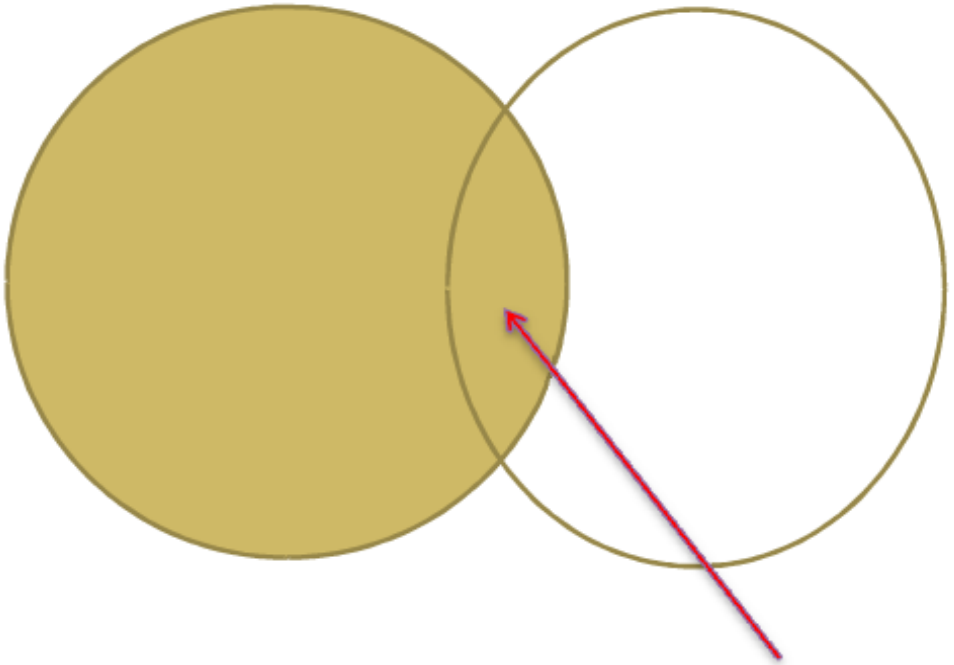
SELECT .... FORM .... WHERE ....

- = <> > < >= <=
- IN , NOT IN (test for several values)
- BETWEEN, NOT BETWEEN (intervals)
- IS NULL, IS NOT NULL
- LIKE, NOT LIKE ( % or _ )
- EXISTS, NOT EXISTS (sub queries)

# Joins

## Joins (INNER)

- Joins are used to combine information from multiple tables
- Basic Syntax of an INNER Join (excludes data that does NOT satisfy the join)



# Joins

- Inner joins are default, so the word 'inner' can be omitted

```
SELECT column_name(s)
FROM table_name1 JOIN table_name2
        ON table_name1.column_name=table_name2.column_name
```

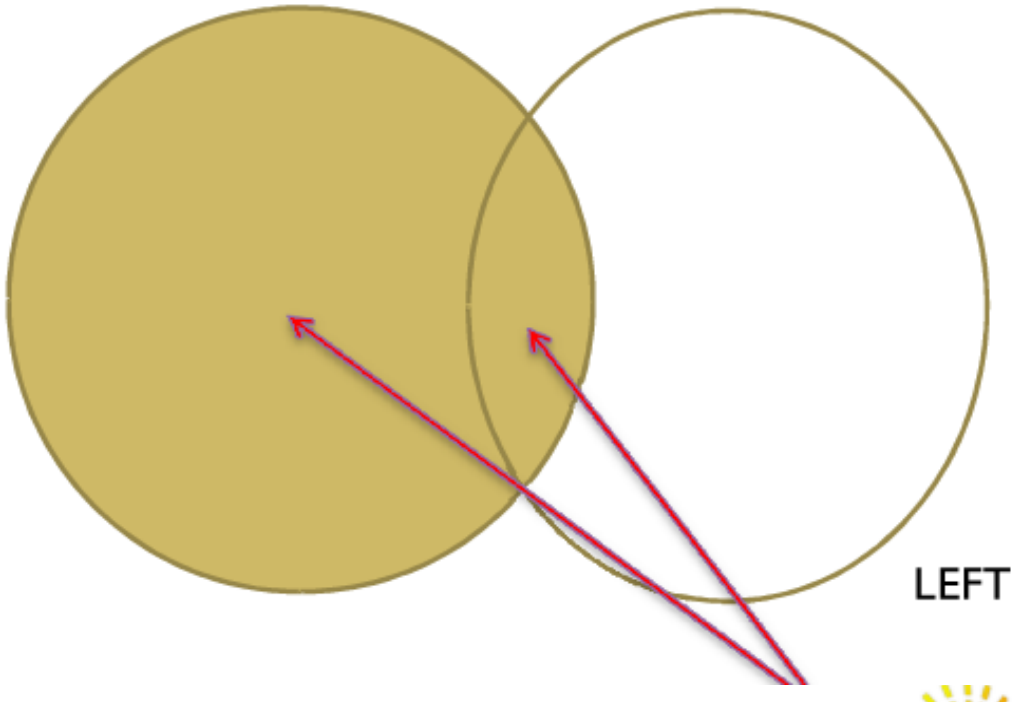
- Natural joins (when foreign key column has the same name as the referenced column)

```
SELECT locus_name, common_name FROM  locus
    JOIN common_name USING (common_name_id);
```

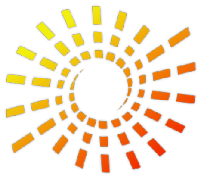
# Joins

## Joins (OUTER)

- Joins are used to combine information from multiple tables
- Basic Syntax of an OUTER Join (include rows from one table that have no matching row)



LEFT OUTER JOIN



# Joins

## Joins (CROSS)

- Each row in one table is paired to every row in the other table
- A cross join is also called a cartesian product

```
SELECT *
FROM   employee CROSS JOIN department
```

| A |
|---|
| B |

| I |
|---|
| 2 |
| 3 |

| A | I |
|---|---|
| A | 2 |
| A | 3 |
| B | I |
| B | 2 |
| B | 3 |

sol genomics network

# Joins

## Joins (OUTER)

table1 LEFT OUTER JOIN table2 – all rows from table 1 will be included

table1 RIGHT OUTER JOIN table2 – all rows from table 2 will be included

table1 FULL OUTER JOIN table2 – all rows from each table will be included

```
SELECT column_name(s)
FROM table_name1
LEFT OUTER JOIN table_name2
ON table_name1.column_name=table_name2.column_name
```

sol genomics network

# Row functions

- Row functions take input parameters and return a value
  - Math Functions
  - String Functions
  - Date and Time Functions
  - Data Type Conversions
  - CASE Statements
  - NULL Functions

# Row functions

Math functions http://www.postgresql.org/docs/8.3/interactive/functions-math.html

```
cxgn=# select round(120.9994, 3) ;
  round
---------
 120.999
(1 row)
```

String functions http://www.postgresql.org/docs/8.3/interactive/functions-string.html

```
cxgn=# select lower ('ABCD') ;
 lower
-------
 abcd
(1 row)
```

sol genomics network

# Row functions

Date and time http://www.postgresql.org/docs/8.1/interactive/functions-datetime.html

```
cxgn=# select now();
              now
-------------------------------
 2009-05-29 04:02:18.043641-04
(1 row)
```

Data type conversion http://www.postgresql.org/docs/8.3/static/sql-createcast.html

# select now(); this is a timestamp!

# select cast (now() as text) ; the output is now a 'text' data type

# Row functions - Case

The SQL CASE expression is a generic conditional expression, similar to if/else statements in other languages:

```
CASE WHEN condition THEN
result
    [WHEN ...]
    [ELSE result]
END
```

```
SELECT * FROM test;

 a
---
 1
 2
 3


SELECT a,
     CASE WHEN a=1 THEN 'one'
        WHEN a=2 THEN 'two'
        ELSE 'other'
     END
   FROM test;

 a | case
---+-------
 1 | one
 2 | two
 3 | other
```

# Row functions –
## more conditional expressions

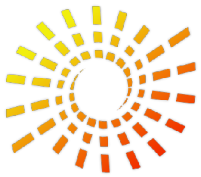
The **COALESCE** function returns the first of its arguments that is not null. Null is returned only if all arguments are null. It is often used to substitute a default value for null values when data is retrieved for display, for example:

SELECT COALESCE(description, short_description, '(none)') ...

The **NULLIF** function returns a null value if value1 and value2 are equal; otherwise it returns value1. This can be used to perform the inverse operation of the COALESCE example given above:

SELECT NULLIF(value, '(none)') ...

If value1 is (none), return a null, otherwise return value1.



# Aggregate functions

- SUM AVG MAX MIN COUNT

```
SELECT AVG(UnitPrice * Quantity) As AveragePrice
FROM WidgetOrders
WHERE Continent = "North America"

SELECT COUNT(*) AS 'Number of Large Orders'
FROM WidgetOrders
WHERE Quantity > 100

SELECT MAX(Quantity * UnitPrice)As 'Largest Order'
FROM WidgetOrders
```

# Aggregate functions - grouping

**GROUP BY** will condense into a single row all selected rows that share the same values for the grouped expressions

```
SELECT count(locus_id) AS loci, common_name_id
FROM locus
GROUP BY common_name_id
ORDER BY common_name_id;

 loci | common_name_id
------+----------------
 2216 |              1
 1038 |              2
  425 |              3
  112 |              4
   48 |              5
  356 |              6
    1 |              7
 1535 |              8
    3 |             32
   46 |             33
    1 |             36
(11 rows)
```

# Aggregate functions - grouping

**HAVING** clause – use with aggregate functions instead of 'WHERE'

```
SELECT count(locus_id) AS loci, common_name_id
FROM locus
GROUP BY common_name_id
HAVING common_name_id > 10;
 loci | common_name_id
------+----------------
   46 |             33
    1 |             36
    3 |             32
(3 rows)
```

sol genomics network

# Subqueries

- A SELECT statement embedded into another statement is called a subquery

```
SELECT SUM(Sales) FROM Store_Information
WHERE Store_name IN
(SELECT store_name FROM Geography
WHERE region_name = 'West')
```

- IN or NOT IN will handle extra lists returned by the sub query

- Operators (>, <, =, etc.) can be used when single values are returned

# Subqueries

```
cxgn=# SELECT locus_name FROM locus
WHERE common_name_id IN
(SELECT common_name_id FROM sgn.common_name WHERE common_name ilike '%tomato%')
LIMIT 10;
                    locus_name
-------------------------------------------
 superba
 praeclusa-2
 xyloglucan endotransglucosylase-hydrolase 8
 xyloglucan endotransglucosylase-hydrolase 10
 xyloglucan endotransglucosylase-hydrolase 12
 xyloglucan endotransglucosylase-hydrolase 13
 xyloglucan endotransglucosylase-hydrolase 23
 deformis2
 Santalene and Bergamotene Synthase
 Z,Z-farnesyl pyrophosphate synthase
(10 rows)
```

# Views

A view is a virtual table based on the results of an SQL statement

http://www.postgresql.org/docs/8.3/interactive/tutorial-views.html

```
CREATE VIEW myview AS
    SELECT city, temp_lo, temp_hi, prcp, date, location
        FROM weather, cities
        WHERE city = name;
```

- A view does not store data
- If you do insert, update, delete on a view the data values in the underlying tables will be updated
  - This is not possible on all views
  - Computed columns cannot be updated
  - Not permitted on aggregate views or views with set operators
  - Join views may or may not be updateable

# postgreSQL

$ psql -h hostname -U username dbname

Production: cxgn (db.sgn.cornell.edu)

Test database: sandbox (db-devel.sgn.cornell.edu)

On your laptops: sandbox (localhost)

Users: postgres, web_usr, your_user_name

Cornell University

sol genomics network

# postgreSQL 8.1

$ psql -h hostname -U username dbname

```
naama@virtual:/data/local/cxgn/core$ psql -h localhost -U web_usr cxgn
Password for user web_usr:
Welcome to psql 8.3.6, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit

SSL connection (cipher: DHE-RSA-AES256-SHA, bits: 256)

cxgn=>
```

Cornell University

sol genomics network

# postgreSQL

Programming with a procedural language
Write your own postgreSQL functions!

- PL/pgSQL
  (similar to Oracle's PL/SQL
  http://en.wikipedia.org/wiki/PL_SQL )
- Many other languages:
  - PL/Perl
    http://www.postgresql.org/docs/8.1/static/plperl.html
  - PL/Java , plPHP, PL/Python, PL/R and more...

Cornell University

sol genomics network

# Connect to the database with Perl

See
$ perldoc DBI

and

$ perldoc CXGN::DB::Connection

```
my  $dbh= CXGN::DB::Connection->new();
$dbh->do("INSERT INTO .... .....");

my $query= "SELECT id, name FROM .....WHERE date = ? ";

my  $sth=$dbh->prepare($query);

$sth->execute($date);

while  (my ($id, $name) = $sth->fetchrow_array()  ) {
        .
        .
}
```

# Resources

- http://solgenomics.net/data/schemas/
- http://internal.sgn.cornell.edu/SGNWiki/PostgresHints
- http://en.wikipedia.org/wiki/SQL
- http://www.postgresql.org/docs/8.3/