An Intro to SQL

Ani Thakar
JHU Institute for Data Intensive Engineering and Science (IDIES)
LSSTC-DSFP Workshop, March 2019

Structured Query Language

- SQL ("sequel") is the lingua franca of the database world
- Allows you to ask sophisticated questions of a database
- There are several ways to query an online database
  - GUIs (some astronomy sites support these)
  - WebForms: simple to complex (we'll see examples)
  - Visual tools, including visual query tools
- SQL is the most powerful way to query a database
- If you expect to do serious research from astro databases, the sooner you learn SQL the better
- We will be using T-SQL, but other flavors are very similar

Parts of a SQL Query

- **Basic SQL query is of the form:**

```
SELECT <data columns> -- comma separated list
FROM <table(s)>
WHERE <search condition>
```

- **The `SELECT` is the only mandatory part, e.g.**

```
"SELECT 1" or "SELECT 'a'" or "SELECT 1.25"
```

  are all valid queries

- **The `FROM` specifies which data table(s) you want**
- **The `WHERE` is the condition the data rows must meet**

SQL Syntax Basics

- SQL is not case-sensitive, and queries can span multiple lines
- Strings are enclosed in single quotes, e.g. `'abcde'`
- Comments:
  - Single line comments start with two dashes and can begin anywhere: `-- From here to end of line ignored`
  - Multi-line comments `/* are specified like this */`
- Equality and inequality operators: = and !=

SQL Don’ts

- If you include a `FROM` clause, never omit the `WHERE` clause, e.g., never do:

```
SELECT x, y
FROM Coords  -- where Coords is a table in a database
```

  unless you know how big the table Coords is, because you are effectively asking for all the rows
- Avoid doing “`SELECT *`” queries as far as possible
  - If you don’t know how many columns the table has
  - These queries are not easy to optimize (will run slower)
- Don’t assume that you can cancel big queries

Simple SQL Queries

- **Basic SELECT-FROM**

```
SELECT TOP 10       -- “TOP x” VERY IMPORTANT!
    objID, ra ,dec -- Get the unique object ID and coordinates
FROM
    PhotoObjAll   -- Table containing all photometric data
```

- **Basic SELECT-FROM-WHERE**

```
SELECT TOP 100
    objID, ra ,dec -- Get the unique object ID and coordinates
FROM
    PhotoPrimary  -- View of PhotoObjAll for unique objects
WHERE
    ra > 185 and ra < 185.1
    AND dec > 15 and dec < 15.1 -- that matches our criteria
```

Using Multiple Tables - JOINs

- A `JOIN` is a way to combine data from multiple tables
- `INNER` (default) `JOIN`
  - Mandatory match between the join columns
  - Rows that don't match will be left out
- `LEFT|RIGHT OUTER JOIN`
  - Inclusive join between two columns
  - Rows that don't match included, with `NULL` values
  - `LEFT` or `RIGHT` depending on which non-match you want to include

JOIN Syntax

- **Only use explicit JOIN syntax**

```
SELECT
      a.key, b.key, a.col1, b.col1, …
FROM table1 as a
      JOIN table2 as b ON a.key=b.key
```

- **NEVER use implicit join syntax (comma-sep. tables)**

```
SELECT
      a.key, b.key, a.col1, b.col1, …
FROM table1 as a, table2 as b
WHERE
      a.key=b.key
```

- **You may forget to specify join condition in long query**

Aggregation and Built-In Functions

- **GROUP BY, ORDER BY, COUNT()**

```
SELECT class, COUNT(*)
FROM SpecObj
GROUP BY class
ORDER BY class          -- optional

SELECT plate.programname, class,
    COUNT(specObjId) AS numObjs
FROM SpecObjAll
    JOIN PlateX AS plate
          ON plate.plate = specObjAll.plate
GROUP BY plate.programname, class
ORDER BY plate.programname, class
```

Some Good Practices

- **Always first estimate the size of the query**
  - **You can do a "`TOP x`" query**
    - **Will get you a sample of the data, e.g., # of *columns* in table**

```
SELECT TOP 10 * FROM SpecObj
```

  - **You can do a `COUNT(*)` query**
    - **If even the `COUNT` takes a long time, it's a biiiig table!**
    - **Will get you the size of the table, i.e., # of *rows* in table**

```
SELECT COUNT(*)FROM SpecObj
```

  - **If the service allows it, try getting a query plan**
- **Remember, you may be drinking from a fire-hose!**

CROSS APPLY and CROSS JOIN

- Advanced `JOIN` versions
- Only attempt if you know what you're doing!
- `CROSS APPLY` applies an operation to every row
  - A version that includes non-matching rows is also available: `OUTER APPLY`
- `CROSS JOIN` is the cartesian product of the rows
  - The same as what would happen if you used implicit join and forgot to specify the join condition

Nested Queries and SET Operations

- You can nest SQL queries
- Substitute a nested query for a table in outer query

```
SELECT x.*
FROM (SELECT …) [AS] x
WHERE
      …
```

- SET ops: `UNION`, `EXCEPT`, `INTERSECT`
  - The number and order of the columns must be the same
  - The data types must be compatible

```
(query1) UNION [ALL]|EXCEPT|INTERSECT (query2) …
```

Functions and Stored Procedures

- Building analysis into the database
- User-Defined Functions return something
  - Scalar functions return a value of a given data type
  - Table-valued functions (TVFs) return a table
  - Accept 0 or more input/output parameters
- Stored Procedures
  - Do not return a value
  - 0 or more input/output parameters
  - Can have optional parameters with default values

Running Queries on SDSS Data

- There are basically 2 modes: sync and async
- SkyServer (Web app) is sync only
- CasJobs (Web app and Web service): sync+async
- You can connect to both via Jupyter using SciServer
  - Please register for an account now if you don't have one
    - http://www.sciserver.org/
  - You need an account to use CasJobs
  - Login optional on SkyServer but there are perks
- All the demos and exercises are based on SDSS data

SkyServer SQL Intro

- Special SkyServer page created for this class
  - http://tinyurl.com/y6538xj8
- 16 sample queries to introduce all the concepts
- You can load and/or run each query with a button
  - If you load it first, you can modify it before running
- Each query introduces something new
- Comments explaining what query does
- Each query is an actual science query

Data Management – Creating Tables

- You can try these out in CasJobs (not SkyServer)
  - Make sure you select the MyDB context first (default)
- Creating a table and inserting data into it:

```
CREATE Table MyFirstTable (
      id INT NOT NULL,
      ra FLOAT NOT NULL,
      dec FLOAT NOT NULL
)
GO
INSERT MyFirstTable VALUES(1, 180, 0.1)
INSERT MyFirstTable VALUES(2, 185, -15.3)
INSERT MyFirstTable VALUES(3, 30, 36.765)
SELECT * FROM MyFirstTable
```

  - Not very convenient or fast way to insert data into a table

Data Management – Inserting Data

- **Inserting data from result of another query (in MyDB context)**

  a) Into an existing table:

```
CREATE Table MySecondTable (
      objID BIGINT NOT NULL,      -- 64-bit INT
      ra FLOAT NOT NULL,    -- double-precision float
      dec FLOAT NOT NULL
)
GO
INSERT MySecondTable
      SELECT TOP 100 objID, ra, dec
      FROM DR15.PhotoTag
```

  b) Into a new table, implicitly creating it

```
SELECT TOP 100 objID, ra, dec
INTO MyThirdTable   -- new table inherits column names
FROM DR15.PhotoTag  -- and types from source table
```

Deleting and Updating Data

- **Deleting a table (in MyDB context)**

```
DROP TABLE MySecondTable
```

  - **But maybe better to use the menu on MyDB page to do it**

- **Deleting some of the data in a table**

```
DELETE MyFirstTable WHERE id = 3
```

- **Updating data in an existing table:**

```
UPDATE MyFirstTable
      SET ra = 189.675
WHERE id = 2
GO
SELECT * FROM MyFirstTable
```

Adding Columns and Creating Indices

- **Adding a column to a table**

```
ALTER TABLE MyThirdTable ADD COLUMN [id] INT NOT NULL
```

- **Creating and dropping a (covering) index**

```
CREATE INDEX ixIdMyThirdTable ON MyThirdTable (id ASC)
INCLUDE (ra, dec)   -- optionally add more columns
GO
DROP INDEX iIdMyThirdTable ON MyThirdTable
```

- **Creating and dropping a clustered index**

```
ALTER TABLE MyFirstTable ADD CONSTRAINT
      pkIdMyFirstTable PRIMARY KEY CLUSTERED (id)
GO
ALTER TABLE MyFirstTable DROP CONSTRAINT
      pkIdMyFirstTable
GO
```

Programming in SQL

(courtesy T. Budavari)

- **Scalar function**

```
CREATE FUNCTION MaxDataX (@ymin float)
RETURNS FLOAT AS
BEGIN
     DECLARE @x float = null;
     SELECT TOP 1 @x = x
     FROM MyData
     WHERE y > @ymin
     ORDER BY X DESC
     RETURN @x;
END
GO
-- Try it out
select dbo.MaxDataX(PI()/10)
select dbo.MaxDataX(99999)
```

- **Table-valued function**

```
CREATE FUNCTION dbo.MaxNDataX(@ymin
float, @n int)
RETURNS @ret TABLE (
     MaxX float not null
) AS
BEGIN
     INSERT @ret SELECT TOP (@n) X
            FROM MyData
            WHERE y > @ymin
     RETURN;
END
GO
-- Try it out
select f.* from dbo.MaxNDataX(-2,5) as
f
```

Exercises

1) Write two versions of a query to get the objID, specObjID, ra, dec, ugriz mags+errors and redshift for the first 1000 objects that have spectra, one without a JOIN and one with a JOIN. You can run it in SkyServer or CasJobs on SDSS DR15.
2) Modify the second version of the query in 1) to include objects that don't have spectra associated with them. Compare the results with 1).
3) Run a Quick query in the CasJobs DR15 context to get the objID, ra, dec for the first 100 imaging objects with ra,dec between (180.0, -0.5) and (181.0, -0.4). Save the result in a MyDB table called MyObjs.
4) Add an int enumerator column called "id" to MyObjs, see the CasJobs FAQ entry for adding an enumerator to a table.
5) Write 2 versions of a query to find the nearest neighbor for each of the objects in MyObjs. One of the versions should use the CROSS APPLY operator, the other version should use just a plain JOIN.
6) Run each of the queries (using the Submit button, not Quick) and save results in tables MyNbrs1 and MyNbrs2 resp.. Compare the contents of the two tables.
7) Create a (max 1000 rows) table in CasJobs for the functions that we created in class
8) Create a "poor man's luminosity function" for objects between z=0.3 and 0.4, using only reliable spectra and excluding invalid magnitudes, and <u>without</u> using a JOIN.