Advanced SQL For Data Science

笔记本：	222-SQL		
创建时间：	9/26/2018 10:28 PM	更新时间：	9/28/2018 5:57 PM

Advanced SQL For Data Science

SQL as a Tool for Data Science

Data Management Operations - What Can SQL DO

- Linking data from different data scources
- Filtering and reformatting data for different uses
- Aggregating data to provide ‘big picture’ summaries
- Answering specfic questions about business operations

Two types of SQL Commands

- **Data Manipulation**
    - Usually about 70%-80% of on a data science project is spent on data manipulation (collecting, preparing, and cleaning data)

```sql
-- Insert
INSERT INTO company_regions
    (region_id, region_name, country)
VALUES
    (1, 'Northeast', 'USA')
```

```sql
-- UPDATE
UPDATE company_regions
SET country='United States'
WHERE country='USA'
```

```sql
-- DELETE
DELETE FROM company_regions
WHERE country='Canada'
```

```
-- SELECT
SELECT * FROM country_regions
```

- **Data Definition**,
  Relational Data Structures:
  - **Table**: Collections of related data records
  - **Indexes**: about the locations of records
  - **Views**: Used when we want to repeatly access to the same set of our data
  - **Schemas**: Collections of tables, indexes, views and other data structures.
  - Schema can have multiple views, and view can have multiple tables, table could include multiple indexes

```
-- Table
Create Table staff(
                    id INTEGER
                    last_name VARCHAR(30)
                    start_data DATE
                    PRIMARY KEY(id)
                    )
```

```
-- Indexes
CREATE INDEX idx_last_name
    ON staff -- Indicate which table will have index
    USING(last_name) -- which columns will be used in the index
```

```
-- View
CREATE VIEW Staff_div AS
    SELECT
        s.id,   -- Aliases, table.column
        s.last_name
        cd.company_division
    From
        staff s
    LEFT JOIN   -- Left join table s and table cd
        company_divisions cd
    ON
        s.department=cd.department
```

```sql
-- Schemas
CREATE SCHEMA data_sci
```

Some Useful Functions & Examples

Data Munging with SQL

- Reformatting character data

```sql
-- create a new column title_dept which can concatenate job_tile to department
Select
    job_title||'-'||department title_dept
From
    staff
```

```sql
-- trim to remove extra spaces
Select
    trim('  Software Engineer  ')
```

```sql
Select
    job_title,(job_title like '%Assistant%') is_asst
From
    Staff
Where
    job_title like 'Assistant%'
```

- Extracting strings from character data

```sql
-- extract the 6th to 8th elements from the whole string
Select
    Substring('abcdefghijkl' From 6 for 3) test_string
```

```
-- Overlay
-- Replacing 1st to 9th string with 'Asst.'

Select
    Overlay(job_title Placing 'Asst.' From 1 For 9)
From
    staff
Where
    job_title LIKE 'Assistant%'
```

- Filtering with regular expressions

```
Select
    job_title
From
    staff
Where
    job_title SIMILAR TO '%Assistant%(III|IV)' -- '%Assistant I_'
        -- '[EPS]%' start with E,P,S
```

- Formating numeric data

```
-- trunc can drop the decimal values without rounding
-- CEIL return the next larger integer
Select
    department, avg(salary), trunc(avg(salary),2), ceil(avg(salary))
From
    staff
Group by
    department
```

Filtering, Joins, and Aggregation

- Subqueries in Select Clauses

```
Select
```

```
    s1.last_name,
    s1.salary,
    s1.department
    (Select round(avg(salary)) From staff s2 Where s2.department=s1.department)
From
    staff s1
```

- Subqueries in From clauses

```
-- Select the salary from people who make more than 100,000 dollars, and subqueries could place in the From clauses
Select
    s1.department,
    round(avg(s1.salary))
From
    (Select
        department,
        salary
    From
        staff
    Where
        salary>100000) s1
Group by
    s1.department
```

- Subqueries in Where clauses

```
    s1.department, s1.last_name,s1.salary
From
    staff s1
Where
    s1.salary=(Select
                    max(s2.salary)
                From
                    staff s2)
```

- Joining tables

```
-- Retrieve data from multiple tables
-- Select null values from two tables which left joined together
Select
    s.last_name,
    s.department,
    cd.company_division
From
    staff s Left Join company_divisions cd
On
    s.department=cd.department
Where
    cd.company_division is null
```

- Creating a view

```
-- Create a view with select and join
Create View staff_div_reg AS
Select
    s.*, cd.company_division,cr.company_regions
From
    staff s
Left Join
    company_divisions cd
On
    s.department=cd.department
Left Join
    company_regions cr
On
    s.region_id=cr.region_id;

Select
    count(*)
From
    staff_div_reg;
```

- Grouping and totaling

```
-- we want to count by both region and division, we can use a feature called grouping sets.
Select
    company_division,
    company_regions,
    gender,
    count(*)
From
    staff_div_reg
Group by
    Grouping Sets(company_division, company_regions,gender)
Order by
    company_regions,company_division,gender
```

- Rollup & Cube to create subtotals

```


Create or Replace View staff_div_reg_country As
    Select
        s.*, cd.company_division, cr.company_regions,cr.country
    From
        staff s
    Left Join
        company_divisions cd
    On
        s.department=cd.department
    Left Join
        company_regions cr
    On
        s.region_id=cr.region_id;

-- Shows each regions in each of two countries
Select
    company_regions,country,count(*)
From
    staff_div_reg_country
Group by
    company_regions, country
Order by
    country,company_regions;

-- Rollup operation on the groupby clause to see the totals from each country
Select
    company_regions,country,count(*)
From
```

```
    staff_div_reg_country
Group by
    Rollup(country,company_regions)
Order by
    country,company_regions;

-- Cube operaiont on the groupby clause to more advanced breakdowns, which shows all combinations of sets of grouping columns
Select
    company_division,company_regions,count(*)
From
    staff_div_reg_country
Group by
    Cube(company_division,company_regions)
```

- Fetch Firest to find top results

```
Select
    last_name, job_title, salary
From
    staff
Order by
    salary DESC
Fetch First
    10 Rows Only;

Select
    company_division, count(*)
From
    staff_div_reg_country
Group by
    company_division
Order by
    count(*) DESC
Fetch First
    5 Rows Only;
```

Window Functions and Ordered Data

Window functions allow us to make SQL statements about rows related to the current rows during processing.

- Over Partition

```
Select
    department, last_name, salary,
    avg(salary) Over (Partition Bydepartment)
From staff
```

- First_Value

```
-- Return the first value based on the sorted order
Select
    department, last_name, salary,
    first_value(salary) Over (Partition by department Order by salary DESC)
From
    staff
```

- Rank

```
-- Rank based on the Order By Function
Select
    department, last_name, salary,
    rank() Over (Partition by department Order by salary DESC)
From
    staff
```

- Lag and Lead

```
-- Lag fuction tell us the rows relative to the currenetly processed rows
Select
    department, last_name, salary,
    Lag(salary) Over (Partition By department Order By salary Desc)
From
    staff;
-- Lead do the opposite of the lag
```

```
Select
    department, last_name, salary,
    Lead(salary,2) Over (Partition By department Order By salary Desc)
From
    staff;
```

- NTILE functions
  NTILE is the window function we use when we want to group rows into some number of buckets or ordered groups

```
Select
    department, last_name, salary,
    ntile(4) Over (Partition By department Order by salary Desc) -- Divide each to 4
 groups based on their salary
From
    staff
```

Preparing Data for Analytics Tools

- Tell Stories Using Data
    - Start with business problem (Losing customers or sales dropping)
    - Stories need data: Pull and prepare data

- Tips for using SQL for Data Science
    - Don’t underestimate teh time needed to collect and prepare data
    - Use aggregate and statistic funcions to understand your data
    - Reformat and check data quality before attempting joins
    - User outer joins to includes as much data as possible
    - Use views to store complet SQL logic
    - Use cubes and rollups for multiple aggregations
    - Use window functions to work with groups of data (replacing subqueries to improve efficiency)