Recently, I have been working on a project related to database and incremental computation for data management, which relates to relational/bag algebra and SQL. Since I had not been exposed to the relevant field before, I quickly went through the basic knowledge related to SQL. And this post is my brief summary while learning the book *Sams Teach Yourself SQL in 10 Minutes, Fourth Edition*.

Excluding *Preliminary*, there are 18 sections in this post:

- In the preliminary section, I record how I set up the environment to run the demos given in the book and show the structures of the tables used as examples in the book. Section 1 lists the basic concepts in SQL that are widely used.
- Next, in Sections 2 - 16, commonly used keywords in SQL are demonstrated with examples and tips.
- Then Section 17 goes a bit further, which contains some interesting but more profound topics related to SQL (links for further reading are given).
- Finally, Section 18 provides some useful websites and materials for learning SQL and further reading.

0 Preliminary

0.1 Set up the Environment Following the Book

1. I tried to download Microsoft SQL Server Express and SQL Server Management Studio, however, it was not easy for a newbee like me to build up a local SQL server :(
2. Therefore, I choose **Oracle Live SQL**. Nothing to download, and everything can be done on the cloud. What users need to do is register for an Oracle a/c.
3. We can download scripts for Oracle Live SQL to generate the tables that will be used for examples.
4. Prepare the tables:

- Open Oracle Live SQL
- `My Script` (sidebar) --> `Upload Script` (upper right corner) --> Upload create.txt & populate.txt
- Run create --> generate the tables
- Run populate --> insert rows for different tables

6. Then we can check the tables by clicking `Schema` (sidebar)
7. Then we can write SQL in `SQL Worksheet` (sidebar), and run the lines by clicking `Run` in the upper right corner

0.2 Structures of the Tables Used as Examples

Table 0.1: Vendor

vend_id	vend_name	vend_address	vend_city	vend_state	vend_zip	vend_country

Table 0.2: Products

prod_id	vend_id	prod_name	prod_price	prod_desc

Table 0.3: Customers

cust_id	cust_name	cust_address	cust_city	cust_state	cust_zip	cust_country	cust_contact	cust_email

Table 0.4: Orders

order_num	order_date	cust_id

Table 0.5: OrderItems

order_num	order_item	prod_id	quantity	item_price

1 Basic Concepts

1. **Database**: A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS).
2. **Table**: A table is an arrangement of information in rows and columns containing cells that make comparing and contrasting information easier.

3. **Schema**: A database schema is considered the “blueprint” of a database which describes how the data may relate to other tables or other data models.
4. **Column**: In a relational database, a column is a set of data values of a particular type, one value for each row of the database.
5. **Row**: In relational databases, a row is a data record within a table.
6. **Primary Key**: A primary key is the column or columns that contain values that uniquely identify each row in a table. A database table must have a primary key for Optim to insert, update, restore, or delete data from a database table.
7. **SQL**: Structured Query Language.
8. **ANSI SQL**: SQL is a popular relational database language first standardized in 1986 by the American National Standards Institute (ANSI). Since then, it has been formally adopted as an International Standard by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).
9. **Keyword**: In SQL, the keywords are the reserved words that are used to perform various operations in the database.
10. **Clause**: Clause in SQL is a built-in function that is used to retrieve the data from the records present in the database.
11. **Search criteria/filter condition**: The search criterion defines the conditions that must be met for an object to be returned by a search query. The criterion consists of a search type and an optional object type. The search type is either parsed string or structured. A parsed string search consists of a list of terms for which to search.
12. **Operator**: An operator is a reserved word or a character that is used to query our database in a SQL expression.
13. **Wildcard**: A wildcard is a character that substitutes for another character or string of characters when searching a database.
14. **Search Pattern**: SQL pattern matching allows you to search for patterns in data if you don't know the exact word or phrase you are seeking. This kind of SQL query uses wildcard characters to match a pattern, rather than specifying it exactly.
15. **Predicate**: A predicate is an expression that evaluates to TRUE, FALSE, or UNKNOWN. Predicates are used in the search condition of WHERE clauses and HAVING clauses, the join conditions of FROM clauses, and other constructs where a Boolean value is required.
16. **Field**: A database field refers to a set of values arranged in a table and has the same data type. A field is also known as a column or attribute. It is not necessary for the values included in a field to be in the form of text alone, as this is not a requirement.
17. **Concatenate**: Concatenation, in the context of databases, refers to the joining together two or more things into a large one. In database parlance, the things being joined are generally two table fields which may be from the same or different tables.
18. **Protable**: Data portability refers to the ability to move, copy or transfer data easily from one database, storage or IT environment to another.
19. **Aggregate function**: An aggregate function performs a calculation on a set of values, and returns a single value. Aggregate functions are often used with the GROUP BY clause of the SELECT statement. All aggregate functions are deterministic.
20. **Query**: a query is simply a request for information. Similarly, the meaning of a query in database management is a request for data.
21. **Relational table**: A relational table is a table of columns or fields that describe a listing (or rows) of data, similar to an Acoustic Campaign database.
22. **Scale**: Scalability is the ability to expand or contract the capacity of system resources in order to support the changing usage of your application. This can refer both to increasing and decreasing usage of the application.
23. **Join**: JOIN is an SQL clause used to query and access data from multiple tables, based on logical relationships between those tables. In other words, JOINS indicate how SQL Server should use data from one table to select the rows from another table.
24. **Union/compound query**: A union operation uses the UNION operator to combine two queries into a single compound query . You can use the UNION operator between two or more SELECT statements to produce a temporary table that contains rows that exist in any or all of the original tables.
25. **View**: In a database, a view is the result set of a stored query on the data, which the database users can query just as they would in a persistent database collection object.

2 SELECT

Examples

```
-- example 1: select the column prod_name from the table Products
SELECT prod_name
FROM Products;

-- example 2: select more than one column (i.e, prod_id and vend_id) from the table Products
SELECT prod_id, vend_id
FROM Products;

-- example 3: based on example 2, only distinct row will be shown due to DISTINCT
SELECT DISTINCT prod_id, vend_id
FROM Products;

-- example 4: the entire table will be shown, since * is a wildcard
SELECT *
FROM Products;
```

Tips

- When selecting multiple columns, separate each column with a comma.
- Using a semicolon to mark the completion of a query.
- As a convention, we capitalize all SQL statements.
- DISTINCT affects all the columns, not only the column following it.

3 Comment

Examples

```
-- example 1: use '--'
SELECT prod_name -- this is a comment
FROM Products;

-- example 2: use '#'
# SELECT prod_name, vend_id
SELECT prod_name
FROM Products;

-- example 3: use '/*' and '*/'
/* SELECT *
FROM Products; */
```

4 ORDER BY

Examples

```
-- example 1: order by according to a single column
SELECT prod_name
FROM Products
ORDERER BY prod_name;

-- example 2: order by according to multiple columns in a sequence
SELECT prod_id, prod_price, prod_name
FROM Prodcuts
ORDER BY prod_price, prod_name;

-- example 3: order by in a descending order
SELECT prod_name
FROM Products
ORDER BY prod_name DESC

-- example 4: order by in an ascending order (not useful, since it is a default setting)
SELECT prod_name
FROM Products
ORDER BY prod_name ASC
```

Tips

- ORDER BY should be placed at the end of the query.
- If we want to order multiple columns in the descending order, we should put DESC after each of them.

5 WHERE & Wildcard

Examples

```
-- example 1: AND
SELECT prod_id, prod_price, prod_name
FROM Products
WHERE vend_id = 'DLL01' AND prod_price <= 4;

-- example 2: OR
SELECT prod_id, prod_price, prod_name
FROM Products
```

```
WHERE vend_id = 'DLL01' OR vend_id = 'BRS01';

-- example 3: % (can represent multiple letters) with [] (s set of characters)
SELECT cust_contact
FROM Customers
WHERE cust_contact LIKE '[JM]%'

-- example 4: _ (can represent a single letter only)
SELECT cust_contact
FROM Customers
WHERE cust_contact LIKE '_J%' -- the second letter is J
```


Tips

- AND has higher priority than OR
- NOT can be used to deney the keyword after it
- Try not to put the wildcard at the begining of a string
- % cannot be used to search for NULL

:fire: 6 Create a Field

Examples


```
-- example 1: concatenate the strings
SELECT RTRIM(vend_name) || '(' || RTRIM(vend_country) || ')'
FROM Vendors
ORDER BY vend_name;
```



```
-- example 2: do some calculation and give the column an alias
SELECT prod_id,
       quantity,
       item_price,
       quantity*item_price AS expanded_price
FROM OrderItems
WHERE order_num = 20008;
```


Tips

- Similar to RTRIM(), there are commands like LRTRIM() and TRIM()
- The field does not exist in the tables in the database

7 Some Useful Functions

Examples

Table 7.1: Commonly used functions related to string processing.

Function	Notes
LEFT()	return the leftmost character in the whole string
LENGTH()	return to the length of the string
LOWER()	convert the whole string into the lowercase
SUBSTRING()	extract part of the string
SOUNDEX()	find the strings according to their pronunciation
UPPER()	convert the whole string into the uppercase

Table 7.2: Commonly used functions related to numerical processing.

Function	Notes
ABS()	return to the absolute value of the number
COS()	return to the consine value of the given degree

Function	Notes
EXP()	return to the exponent value of the given number
PI()	return to the pi value
SIN()	return to the sine value of the given degree
SQRT()	return to the square root of the given number
TAN()	return to the tangent value of the given degree

8 Aggregate

Examples

Table 8.1: Commonly used SQL aggregate functions.

Functions	Notes
AVG()	return to the average of a selected column
COUNT()	return the number of rows of a selected column
MAX()	return to the maximal value of a selected column
MIN()	return to the minimal value of a selected column
SUM()	return to the sum of a selected column

Tips

- DISTINCT cannot be used with COUNT(*)
- It is meaningless to use DISTINCT with MIN() or MAX(), although it is doable

9 GROUP BY

Examples

```
-- example 1: use GROUP BY only
SELECT vend_id, COUNT(*) AS num_prods
FROM Products
GROUP BY vend_id;

-- example 2: use GROUP BY with HAVING
SELECT cust_id, COUNT(*) AS orders
FROM Orders
GROUP BY cust_id
HAVING COUNT(*) >= 2;
```


Tips

- WHERE conducts filtering on every single row
- HAVING conducts filtering on the combination of multiple rows
- Without GROUP BY (each row is a single group), HAVING and WHERE play the same function
- ORDER BY should be placed after GROUP BY, and always at the end of a query
- From top to bottom, the order of the keywords should be: SELECT --> FROM --> WHERE --> GROUP BY --> HAVING --> ORDER BY

10 Subquery

Examples

```
-- example 1:
/* Explanation: We want to find the customers' information who have ordered the product with ID 'RGAN01'.
The 2nd SELECT after WHERE provides order_num containing the products with ID 'RGAN01'.
Then we use the returned order_num from OrderItems to find the available cust_id in Orders.
```

```
Finally, we use the cust_id from Orders to find customer information in Customers. */
SELECT cust_name, cust_contact
FROM Customers
WHERE cust_id IN (SELECT cust_id
                  FROM Orders
                  WHERE order_num IN (SELECT order_num
                                      FROM OrderItems
                                      WHERE prod_id ='RGAN01'));


-- example 2:
/* Explanation: We want to find the customers' IDs who have order items with values beyond 10.
The 2nd SELECT after WHERE provides the order_num containing the item with a price higher than 10.
Then we use the returned order_num to help us find the cust_id in orders.
Finally, we use the returned cust_id from Orders to find the ID we want. */
SELECT cust_id
FROM Customers
WHERE cust_id IN (SELECT cust_id
                  FROM Orders
                  WHERE order_num IN (SELECT order_num
                                      FROM OrderItems
                                      WHERE item_price > 10));

-- example 3:
/* Explanation: We want to find the customers' IDs and the total money spent on their orders.
The 2nd SELECT uses aggregation function SUM() to generate a new field using the columns in table OrderItems with alias total_money. */
SELECT cust_id
      (SELECT SUM(order_item*item_price)
       FROM OrderItems
       WHERE Orders.order_num = OrderItems.order_num) AS total_money
FROM Orders
ORDER BY total_money;
```

Tips

- subquery can be replaced by JOIN operation
- subquery can work with WHERE to filter the selected rows (cf. examples 1 and 2)
- subquery allows obtaining resutls from more than one tables (cf. example 3)

11 JOIN

```
-- example 1: use where to join the tables
SELECT vend_name, prod_name, prod_price
FROM Vendors, Products
WHERE Vendors.vend_id = Products.vend_id

-- example 2: INNER JOIN
SELECT vend_name, prod_name, prod_price
FROM Vendors
INNER JOIN Products ON Vendors.vend_id = Products.vend_id;

-- example 3: use table alias when doing join
SELECT cust_name, cust_contact
FROM Customers AS C, Orders AS O, OrderItems AS OI
WHERE C.cust_id = O.cust_id
      AND OI.order_num = O.order_num
      AND prod_id = 'RGAN01';

-- example 4: LEFT OUTER JOIN
SELECT Customers.cust_id, Orders.order_num
FROM Customers
LEFT OUTER JOIN Orders ON Customers.cust_id = Orders.cust_id;

-- example 5: RIGHT OUTER JOIN
SELECT Customers.cust_id, Orders.order_num
FROM Customers
RIGHT OUTER JOIN Orders ON Customers.cust_id = Orders.cust_id;

-- example 6: FULL OUT JOIN
SELECT Cusotmers.cust_id, Orders.order_num
FROM Customers
FULL OUTER JOIN Orders ON Customers.cust_id = Orders.cust_id;
```

```sql
-- example 7: JOIN with aggregate functions
SELECT Customers.cust_id,
       COUNT(Orders.order_num) AS num_ord
FROM Cusomers
INNER JOIN Orders ON Customers.cust_id = Orders.cust_id
GROUP BY Customers.cust_id;
```

Tips

- When using OUTER JOIN, we must make it specific that whether we use LEFT or RIGHT OUTER JOIN
- LEFT OUTER JOIN: every row in the table mentioned above will be reserved
- RIGHT OUTER JOIN: every row in the table mentioned after the join command will be reserved
- We are always expected to give the condition when doing join, or the tables will do Cartesian product, which is costly.

12 UNION

Examples

```sql
-- exmple 1: UNION without repeated rows
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_state IN ('IL', 'IN', 'MI')
UNION
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_name = 'Fun4All'

-- example 2: UNION ALL with repeated rows
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_state IN ('IL', 'IN', 'MI')
UNION ALL
SELECT cust_name, cust_contact, cust_email
FROM Customers
WHERE cust_name = 'Fun4All'
```

Tips

- UNION can be used only when there are more than one SELECT
- The column names from different tables can be different, but the names of the output columns will follow the first table
- UNION will delete the repeated rows automatically, while UNION ALL will reserve all of them

13 INSERT INTO

Examples

```sql
-- example 1: insert a complete row in a simple will (NOT recommended)
INSERT INTO Customers
VALUES(1000000006,
       'Toy Land',
       '123 Any Street',
       'New York',
       'NY',
       '11111',
       'USA',
       NULL,
       NULL);

-- example 2: insert a complete row in a complex but safe way (recommended)
INSERT INTO Customers(cust_id,
                      cust_name,
                      cust_address,
                      cust_city,
                      cust_state,
                      cust_zip,
                      cust_country,
```

```
                    cust_contact,
                    cust_email)
VALUES(1000000006,
       'Toy Land',
       '123 Any Street',
       'New York',
       'NY',
       '11111',
       'USA',
       NULL,
       NULL);

-- example 3: insert a partial row
INSERT INTO Customers(cust_id,
                    cust_name,
                    cust_address,
                    cust_country,
                    cust_contact,
                    cust_email)
VALUES(1000000006,
       'Toy Land',
       '123 Any Street',
       'New York',
       NULL,
       NULL);

-- example 4: insert a selected row
INSERT INTO Customers(cust_id,
                      cust_name,
                      cust_address,
                      cust_city,
                      cust_state,
                      cust_zip,
                      cust_country,
                      cust_contact,
                      cust_email)
SELECT cust_id,
       cust_name,
       cust_address,
       cust_city,
       cust_state,
       cust_zip,
       cust_country,
       cust_contact,
       cust_email)
FROM CustNew;

-- example 5: copy a table
CREATE TABLE CustCopy AS SELECT * FROM Customers;
```

Tips

- In example 4, the columns' names after SELECT are not important, the values are inserted into the table according to their positions.
- INSERT INTO usually inserts one row a time, but INSERT SELECT can insert multiple rows based on the number of rows extracted.

14 UPDATE/DELETE

Examples

```
-- example 1: UPDATE
UPDATE Customers
SET cust_contact = 'Sam Roberts'
    cust_email = 'sam@toyland.com
WHERE cust_id = 1000000006

-- example 2: DELETE
DELETE FROM Customers
WHERE cust_id = 1000000006
```

Tips

- We are expected to use WHERE after UPDATE, or all rows will be updated

- We can use UPDATE to delete selected columns by setting their values to NULL
- DELETE cannot delete the table itself, although it can delete all rows in this table
- We always add FROM after DELETE

15 CREATE/ALERT/DROP TABLE

Examples


```
-- example 1: CREATE TABLE
CREATE TABLE OrderItems
(
    order_num   INTEGER   NOT NULL,
    order_item  INTEGER   NOT NULL,
    prod_id     CHAR(10)  NOT NULL,
    quantity    INTEGER   NOT NULL    DEFAULT 1,
    item_price  DECIMAL(8,2)    NOT NULL
);
```



```
-- example 2: add a column in an existing table
ALTER TABLE Vendors
ADD vend_phone CHAR(20);
```



```
-- example 3: delete a column in an existing table
ALTER TABLE Vendors
DROP COLUMN vend_phone;

-- example 4: delete a table
DROP TABLE Vendor;
```


Tips

- There will not be a confirmation step before deleting a table or an undo step after deleting a table, so we should be very careful when we use DROP TABLE

16 VIEW

Examples


```
-- example 1: create a view
CREATE VIEW OrderItemsExpanded AS
SELECT order_num,
       prod_id,
       quantity,
       item_price,
       quantitiy*item_price AS expanded_price
FROM OrderItems;
```



```
-- example 2: use the created view
SELECT *
FROM OrderItemsExpanded
WHERE order_num = 20008;
```



```
-- example 3: delete the view
DROP VIEW OrderItemsExpanded;
```


Tips

- VIEW is a query, and itself does not contain any data
- If we use multiple JOINs and filters to create a view, and use this view in another view. This operation will be extremely costly.

17 Let's Go a Bit Further

- Working with Stored Procedures [Blog]
- Managing Transaction Processing [Definition]
- Using Cursors [Tutorial]

18 Useful Learning Materials

- Oracle University
- IBM DB2 SQL Workshop
- W3 School