SQL (1)

William Wang

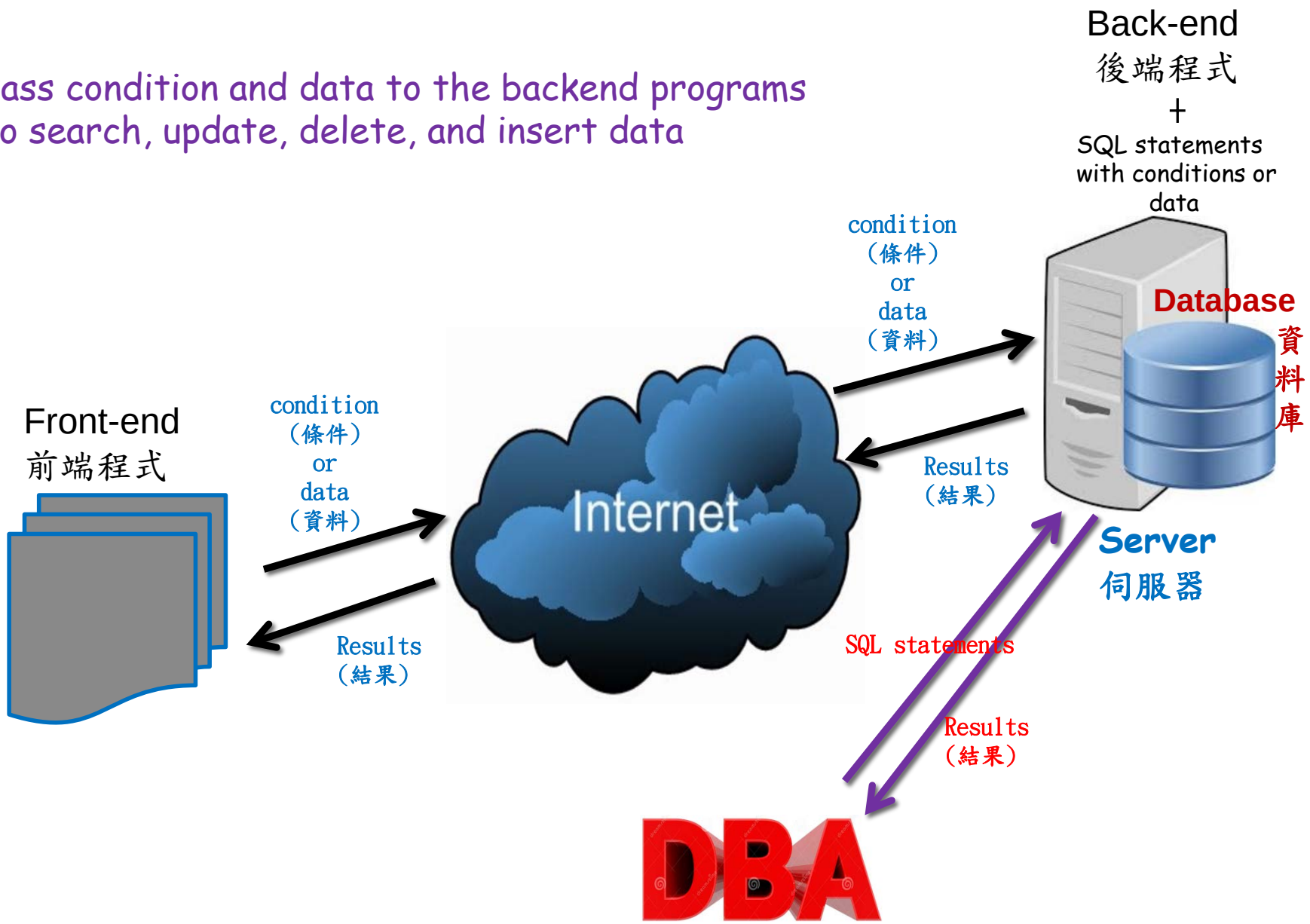
Contents

- Why SQL statements
- SQL statements
- Table Joins
- Other functions used in SQL statements
- Varchar *v.s.* Char
- Primary key and Foreign key
- Types of Databases
- RDB

Why SQL Statements

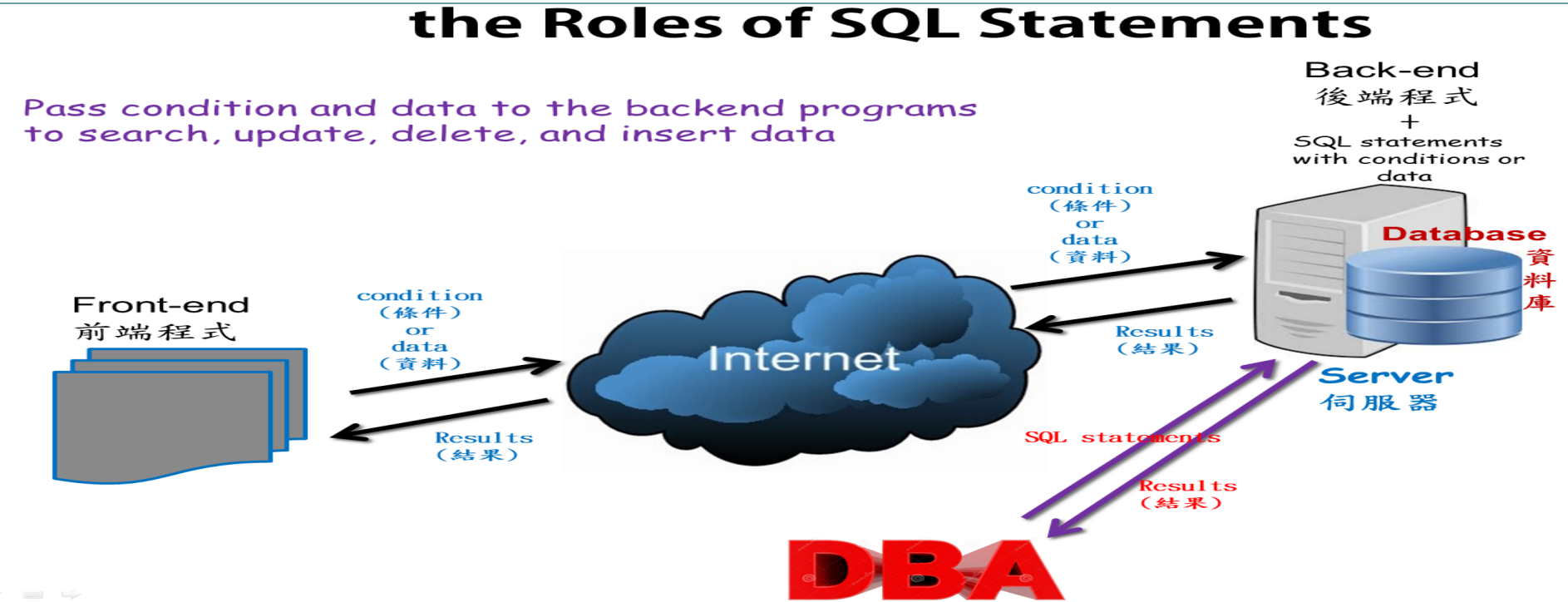
the Roles of SQL Statements

Pass condition and data to the backend programs to search, update, delete, and insert data

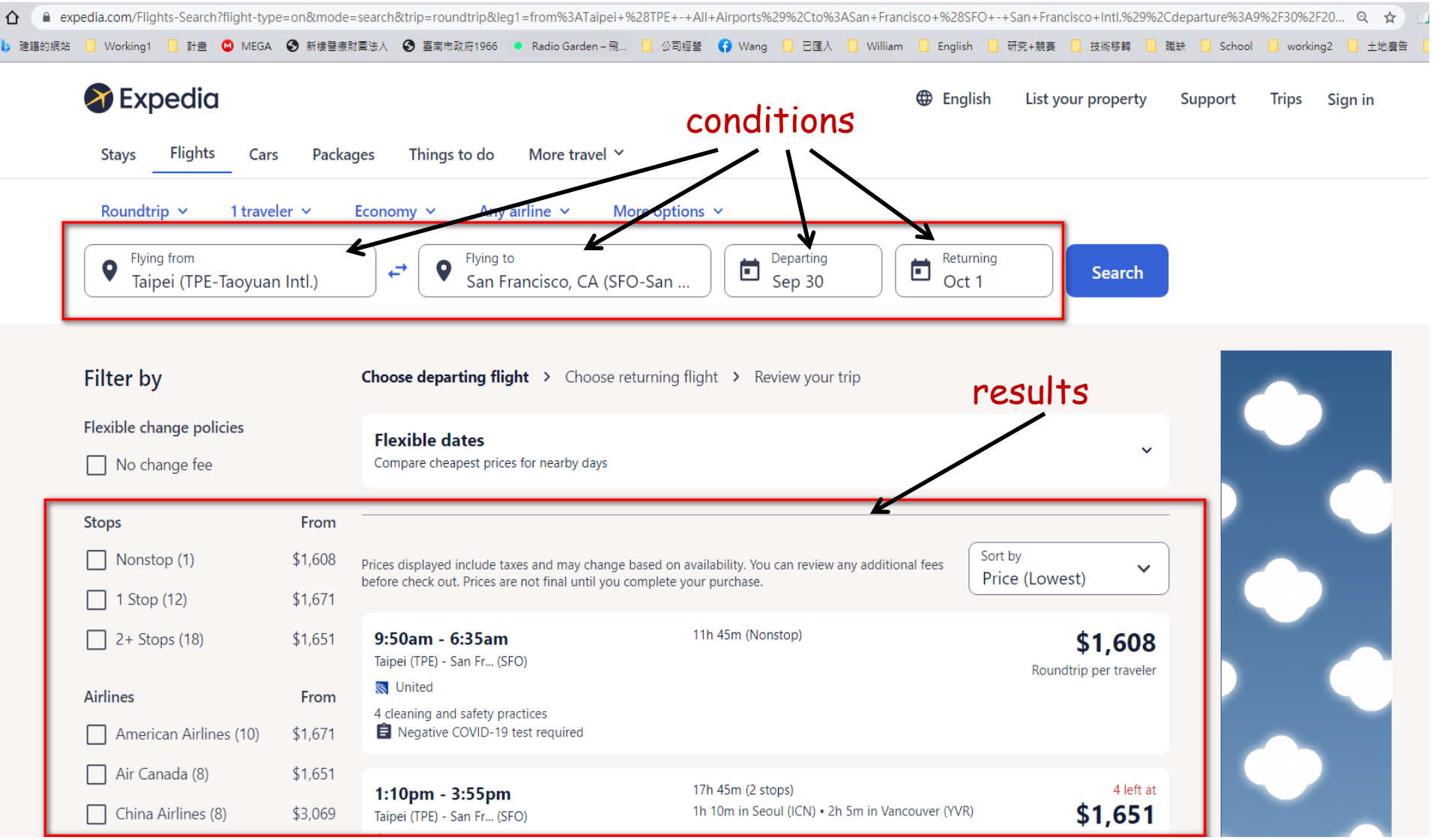


SQL – DML(Data Manipulation Language)

- DML has 4 basic data manipulation
  - Search data with condition
  - Update data with condition
  - Delete data with condition
  - Insert data with condition



Air Ticket Booking System



SQL

- Structured Query Language, commonly known as SQL, is a standard programming language for relational databases. Despite being older than many other types of code, it is the most widely implemented database language.
- Structured Query Language:為結構化查詢語言. 是一種特定目的程式語言，用於管理與存取關聯式資料庫管理系統（RDBMS）
- SQL is pronounced **"sequel"** (/ˈsiːkwəl/).

SQL Statements

https://dev.mysql.com/doc/refman/8.0/en/dynindex-statement.html
https://www.1keydata.com/tw/sql/sql-length.html

SQL Database Schema

- We define SQL Schema as **a logical collection of database objects**. A user owns that owns the schema is known as schema owner. It is a useful mechanism to segregate database objects for different applications, access rights, managing the security administration of databases. (在資料庫內資料物件(含資料表)的設計))

➔ Please import ksu_db0914.sql

SQL SELECT

- Simple Syntax:

  SELECT “column names" FROM “table names";

```
SELECT Store_Name FROM Store_Information;
```

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Store_Name
Los Angeles
San Diego
Los Angeles
Boston

SELECT DISTINCT

- Simple Syntax:

  SELECT DISTINCT “column names“ FROM “table names";

```
SELECT DISTINCT Store_Name FROM Store_Information;
```

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Store_Name
Los Angeles
San Diego
Boston

SQL WHERE

- Simple syntax:

SELECT “column names“ FROM “table names“ WHERE “conditions";

```
SELECT Store_Name
FROM Store_Information
WHERE Sales > 1000;
```

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Store_Name
Los Angeles

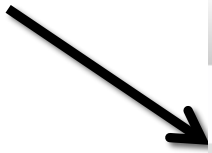
Copy ?

. Copy the structure and data of table Store_Information to a new table Store_Information_1

How ?

Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08



Store_Information_1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Update data

```
UPDATE store_information SET Store_Name = "San Francisco" WHERE Sales = 300;
```

Store_Name	Sales	Txn_Date
Los Angeles	1500	2019-10-05
San Diego	250	2019-10-07
San Francisco	300	2019-10-08
Boston	700	2019-10-08

Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
San Francisco	300	1999-01-08
Boston	700	1999-01-08

SQL AND OR

- **Simple syntax:**

SELECT “column names“ FROM “table names“ WHERE
(“condition“ [AND|OR] “condition“)+;

```
SELECT Store_Name
FROM Store_Information
WHERE Sales > 1000
OR (Sales < 500 AND Sales > 275);
```

Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
San Francisco	300	1999-01-08
Boston	700	1999-01-08

Store_Name
Los Angeles
San Francisco

SQL IN

- Simple syntax:

SELECT “column names“ FROM “table names“
WHERE “column name" IN (‘value 1', ‘value2', ...);

```
SELECT *
FROM Store_Information
WHERE Store_Name IN ('Los Angeles', 'San Diego');
```

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
San Francisco	300	1999-01-08
Boston	700	1999-01-08

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07

SQL Between

- Simple syntax:

SELECT “column names“ FROM “table names"
WHERE “column name" BETWEEN ‘value 1' AND ‘value 2';

```
SELECT * FROM Store_Information WHERE Txn_Date BETWEEN '1999-01-06' AND '1999-01-10';
```

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
San Francisco	300	1999-01-08
Boston	700	1999-01-08

Store_Name	Sales	Txn_Date
San Diego	250	1999-01-07
San Francisco	300	1999-01-08
Boston	700	1999-01-08

Wildcard

- Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a__%'	Finds any values that starts with "a" and are at least 2 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

SQL LIKE

- Simple syntax:

SELECT “column names“ FROM “table names“ WHERE “coulmn name" LIKE “pattern”;

```
SELECT *
FROM Store_Information
WHERE store_name LIKE '%AN%';
```

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
San Francisco	300	1999-01-08
Boston	700	1999-01-08

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
San Francisco	300	1999-01-08

SQL ORDER BY

- Simple syntax:

SELECT “column names“ FROM “table names“

[WHERE “conditions"]
ORDER BY “column name" [ASC, DESC];

```
SELECT Store_Name, Sales, Txn_Date
FROM Store_Information
ORDER BY Sales DESC;
```

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
San Francisco	300	1999-01-08
Boston	700	1999-01-08

Store_Name	Sales ▼ 1	Txn_Date
Los Angeles	1500	1999-01-05
Boston	700	1999-01-08
San Francisco	300	1999-01-08
San Diego	250	1999-01-07

SQL Aggregation Functions

- An aggregate function allows you to perform a calculation on a set of values to return a single scalar value. We often use aggregate functions with the GROUP BY and HAVING clauses of the SELECT statement. The following are the most commonly used SQL aggregate functions:
  - AVG – calculates the average of a set of values.
  - COUNT – counts rows in a specified table or view.
  - MIN – gets the minimum value in a set of values.
  - MAX – gets the maximum value in a set of values.
  - SUM – calculates the sum of values.

SQL AVG

- Simple syntax:
  SELECT AVG(“column name") FROM “table name";

```
SELECT AVG(Sales) FROM Store_Information;
```
1

Store_Information 1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

AVG(Sales)
687.5000

?

AVG(Sales)
687.5

SQL COUNT

- Simple syntax:

  SELECT COUNT(“column name") FROM “table names";

```
SELECT count(Store_Name) FROM store_information_1 WHERE store_name is not null;
```

Store_Information_1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

COUNT (Store_Name)
4

SQL MAX

- Simple syntax:

  SELECT MAX(“column name") FROM “table names";

```
SELECT MAX (Sales) FROM Store_Information; _1
```

Store_Information _1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

MAX (Sales)
1500

SQL MIN

- Simple syntax:

  SELECT MIN(“column name") FROM “table names";

```
SELECT MIN (Sales) FROM Store_Information;_1
```

Store_Information _1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

MIN (Sales)
250

SQL SUM

- Simple syntax:

  SELECT SUM(“column name") FROM “table names";

```
SELECT SUM (Sales) FROM Store_Information;_1
```

Store Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

SUM (Sales)
2750

SQL GROUP BY - 1

- Simple syntax:

  SELECT “column name 1", SUM(“column 2") FROM “tablenames" GROUP BY “column name 1";

```
SELECT Store_Name, SUM(Sales)
FROM Store_Information_1
GROUP BY Store_Name;
```

Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Store_Name	SUM(Sales)
Los Angeles	1800
San Diego	250
Boston	700

SQL GROUP BY - 2

- Simple syntax:

  SELECT “column name 1", SUM(“column 2") FROM “tablenames" GROUP BY “column name 1";

```
SELECT Store_Name, SUM(Sales), Sales
FROM store_information_1
GROUP BY Store_Name;
```

✘

Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Store_Name	SUM(Sales)	Sales
Boston	700	700
Los Angeles	1800	1500
San Diego	250	250

Right syntax + wrong semantic SQL ➔ wrong output

SQL HAVING

- SELECT “column name 1", SUM(“column name 2")
  FROM “table names"
  GROUP BY “column name1“
  HAVING (condition of aggregation function);

```
SELECT Store_Name, SUM(Sales)
FROM Store_Information _1
GROUP BY Store_Name
HAVING SUM(Sales) > 1500;
```

Store_Information_1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Store_Name	SUM(Sales)
Los Angeles	1800

What is the difference between
where clause and having clause?

Alias for table name and column name

- Simple syntax:

  SELECT “tableAlias".“column name“ “column alias"
  FROM “tablename" “tableAlias";

```
SELECT A1.Store_Name Store, SUM(A1.Sales) "Total Sales" FROM Store_Information_1 A1 GROUP BY A1.Store_Name;
```

Store_Information _1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Store	Total Sales
Boston	700
Los Angeles	1800
San Diego	250

Us *As* for Alias usage

- Simple syntax:

  SELECT “tableAlias”.“columnname1” as “columnAlias”
  FROM “table name" “tableAlias";

```
SELECT A1.Store_Name as Store,
       SUM(A1.Sales) as 'Total Sales'
FROM Store_Information_1 as A1
GROUP BY A1.Store_Name;
```

Store_Information 1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Store	Total Sales
Boston	700
Los Angeles	1800
San Diego	250

Table Joins

Table Joins

- Table Joins in MySQL by using
  - where clause. It also can is used with SQL where conditions.
  - no where clause. However, it does not mean you cannot use where clause for SQL where conditions. (Inner join/ Outer join)

Table Join using where - 1 (Before grouping by)

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Store_Information_1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

Geography A1

region_name	store_name
East	Boston
West	Los Angeles
West	Los Angeles
West	San Diego

Store_Information_1 A2

store_name	sales
Boston	700
Los Angeles	1500
Los Angeles	300
San Diego	250

```
SELECT A1.Region_Name REGION, SUM(A2.Sales) SALES
FROM Geography A1, Store_Information_1  A2
WHERE A1.Store_Name = A2.Store_Name
GROUP BY A1.Region_Name;
```

REGION	SALES
East	700
West	2050

Right syntax + right semantic SQL

Table Join using where - 2

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Store_Information_1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

(Before grouping by)

Geography A1		Store_Information_1 A2	
region_name	store_name	store_name	sales
East	Boston	Boston	700
West	Los Angeles	Los Angeles	1500
West	Los Angeles	Los Angeles	300
West	San Diego	San Diego	250

```
SELECT A1.Region_Name REGION, SUM(A2.Sales) SALES, A1.store_name
FROM Geography A1, Store_Information_1  A2
WHERE A1.Store_Name = A2.Store_Name
GROUP BY A1.Region_Name;
```

The logic answer are ok on this case due to where!

REGION	SALES	store_name
East	700	Boston
West	2050	Los Angeles

Right syntax + wrong semantic SQL ➔ possible wrong output

Table Join using where - 3

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Store_Information_1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

(Before grouping by)

Geography A1		Store_Information_1 A2	
region_name	store_name	store_name	sales
East	Boston	Boston	700
West	Los Angeles	Los Angeles	1500
West	Los Angeles	Los Angeles	300
West	San Diego	San Diego	250

```
SELECT A1.Region_Name REGION, SUM(A2.Sales) SALES, A1.store_name
FROM Geography A1, Store_Information_1 A2
WHERE A1.Store_Name = A2.Store_Name
GROUP BY A1.Region_Name, A1.store_name;
```

REGION	SALES	store_name
East	700	Boston
West	1800	Los Angeles
West	250	San Diego

Right syntax + right semantic SQL

Table Join using where - 4

(Before grouping by)

Geography A1

region_name	store_name
East	Boston
West	Los Angeles
West	Los Angeles
West	San Diego

Store_Information_1 A2

store_name	sales
Boston	700
Los Angeles	1500
Los Angeles	300
San Diego	250

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Store_Information_1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08

```
SELECT A1.Region_Name REGION, SUM(A2.Sales) SALES , A1.store_name
FROM Geography A1, Store_Information_1  A2
WHERE (A1.store_name = A2.Store_Name) AND
      (A1.Region_Name like "%W%")
GROUP BY A1.Region_Name, A1.store_name;
```

REGION	SALES	store_name
West	1800	Los Angeles
West	250	San Diego

Right syntax + right semantic SQL

Table Join without **where** clause

- Inner Join - Inner Join clause basically **creates a output by combining rows that have matching values in two tables or more than two tables**. This join is based on a logical relationship (or a common field) between the tables and is used to retrieve data that appears in both tables. ← The inner join matches each row in one table with every row in other tables and allows you to query rows that contain columns from both tables.

- Outer Joins – The SQL outer join **returns all rows from both the participating tables which satisfy the join condition along with rows which do not satisfy the join condition**. ← Outer joins, on the other hand, are for finding records that may not have a match in the other table. As such, you have to specify **which side** of the join is allowed to have a missing record.

- Outer Joins -
  - LEFT JOIN and RIGHT JOIN are shorthand for LEFT OUTER JOIN and RIGHT OUTER JOIN
  - Left Outer Join would get us all the records from the left table regardless of whether or not they have a match in the right table
  - Full Outer join – MySQL does not support full outer join so far! However,…..
- What is the difference between join and where?
  - One difference is that the **first option hides the intent by expressing the join condition in the where clause**. The second option, where the join condition is written out is more clear for the user reading the query. It shows the exact intent of the query. ← statement presentation's problem!

Inner Join - 1

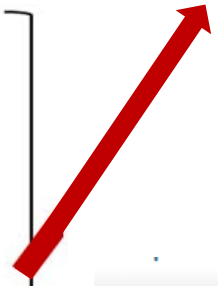
```
SELECT A1.Region_Name, A1.store_name, A0.Store_Name, A0.Sales
FROM Geography A1
INNER JOIN Store_Information  A0
  ON A1.store_name = A0.Store_Name
```

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
San Francisco	300	1999-01-08
Boston	700	1999-01-08



Region_Name	store_name	Store_Name	Sales
West	Los Angeles	Los Angeles	1500
West	San Diego	San Diego	250
East	Boston	Boston	700

Inner Join - 2

- This join has the same answer as the one on page 34 without group by.

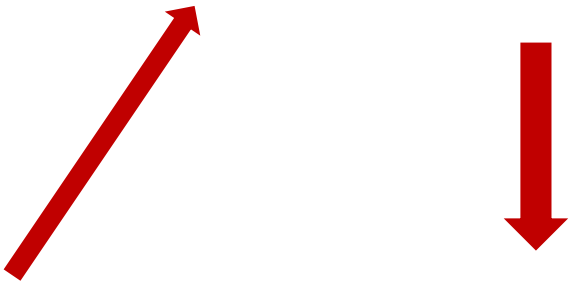
```
SELECT A1.Region_Name, A1.store_name, A3.Store_Name, A3.Sales
FROM Geography A1
INNER JOIN Store_Information_1  A3
  ON A1.store_name = A3.Store_Name
```

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Store_Information_1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08



Region_Name	store_name	Store_Name	Sales
West	Los Angeles	Los Angeles	1500
West	San Diego	San Diego	250
West	Los Angeles	Los Angeles	300
East	Boston	Boston	700

Inner Join

- This join has the same answer as the one on page 34.

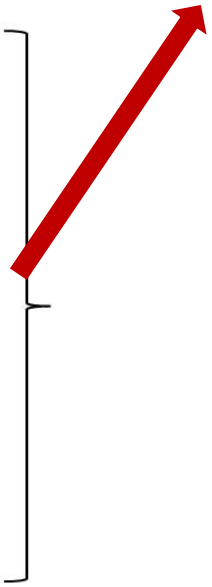
```
SELECT A1.Region_Name REGION, SUM(A2.Sales) SALES
FROM Geography A1
INNER JOIN Store_Information_1  A2
      ON A1.store_name = A2.Store_Name
GROUP BY A1.Region_Name;
```

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Store_Information_1

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
Los Angeles	300	1999-01-08
Boston	700	1999-01-08



REGION	SALES
East	700
West	2050

Left outer Join

Left Outer Join would get us all the records from the left table regardless of whether or not they have a match in the right table

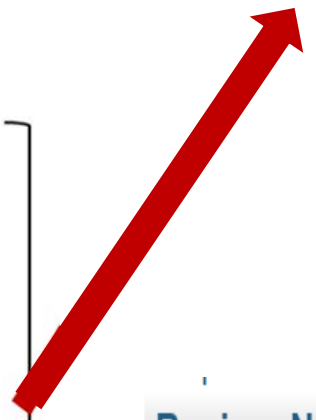
```
SELECT A1.Region_Name, A1.store_name, A5.Store_Name,  A5.Sales
FROM Geography A1
LEFT JOIN Store_Information  A5
     ON A1.store_name = A5.Store_Name
```

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
San Francisco	300	1999-01-08
Boston	700	1999-01-08



Region_Name	store_name	Store_Name	Sales
West	Los Angeles	Los Angeles	1500
West	San Diego	San Diego	250
East	Boston	Boston	700
East	New York	*NULL*	*NULL*

Left outer Join

Right Outer Join would get us all the records from the right table regardless of whether or not they have a match in the left table

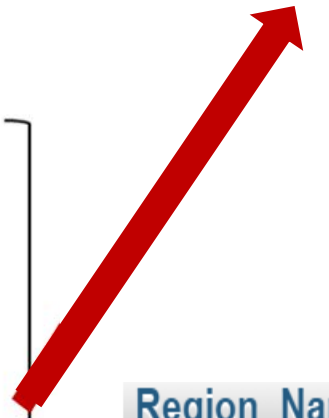
```
SELECT A1.Region_Name, A1.store_name, A5.Store_Name,  A5.Sales
FROM Geography A1
RIGHT JOIN Store_Information  A5
     ON A1.store_name = A5.Store_Name
```

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Store_Information

Store_Name	Sales	Txn_Date
Los Angeles	1500	1999-01-05
San Diego	250	1999-01-07
San Francisco	300	1999-01-08
Boston	700	1999-01-08

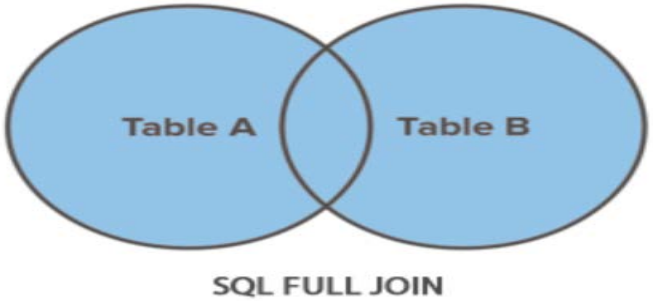


Region_Name	store_name	Store_Name	Sales
East	Boston	Boston	700
West	Los Angeles	Los Angeles	1500
West	San Diego	San Diego	250
NULL	*NULL*	San Francisco	300

Full Outer Join

- MySQL does not support full outer join out of the box, unlike other databases such as PostgreSQL, and SQL Server. So you will need to do a full outer join using a combination of other join types such as LEFT JOIN ad RIGHT JOIN that are supported in MySQL.

```
SELECT * FROM t1
LEFT JOIN t2 ON t1.id = t2.id
UNION ALL
SELECT * FROM t1
RIGHT JOIN t2 ON t1.id = t2.id
WHERE t1.id IS NULL
```



SQL FULL JOIN

- The above query will also return duplicate rows, if any. If you don't want duplicate records in full outer join, use the following query instead.

```
SELECT * FROM t1
LEFT JOIN t2 ON t1.id = t2.id
UNION
SELECT * FROM t1
RIGHT JOIN t2 ON t1.id = t2.id
```

```sql
SELECT A1.Region_Name, A1.store_name, A5.Store_Name,  A5.Sales
FROM Geography A1
LEFT JOIN Store_Information  A5
     ON A1.store_name = A5.Store_Name
UNION
SELECT A1.Region_Name, A1.store_name, A5.Store_Name,  A5.Sales
FROM Geography A1
RIGHT JOIN Store_Information  A5
     ON A1.store_name = A5.Store_Name
```

Region_Name	store_name	Store_Name	Sales
West	Los Angeles	Los Angeles	1500
West	San Diego	San Diego	250
East	Boston	Boston	700
East	New York	*NULL*	*NULL*

Region_Name	store_name	Store_Name	Sales
East	Boston	Boston	700
West	Los Angeles	Los Angeles	1500
West	San Diego	San Diego	250
NULL	*NULL*	San Francisco	300

Region_Name	store_name	Store_Name	Sales
West	Los Angeles	Los Angeles	1500
West	San Diego	San Diego	250
East	Boston	Boston	700
East	New York	*NULL*	*NULL*
NULL	*NULL*	San Francisco	300

Other Functions used in SQL statements

CONCATENATION

- Simple syntax:

  CONCAT(String 1, String 2, String 3, ...)

```
SELECT CONCAT(Region_Name, Store_Name)
FROM Geography
WHERE Store_Name = 'Boston';
```

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

CONCAT(Region_Name, Store_Name)
EastBoston

SUBSTRING

- Simple syntax:

  SUBSTR (string, position)

```
SELECT SUBSTR(Store_Name, 3)
FROM Geography
WHERE Store_Name = 'Los Angeles';
```

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

SUBSTR(Store_Name, 3)
s Angeles

SUBSTRING

- Simple syntax:

  SUBSTR (string, position)

```
SELECT SUBSTR(Store_Name,2,4)
FROM Geography
WHERE Store_Name = 'San Diego';
```

Geography

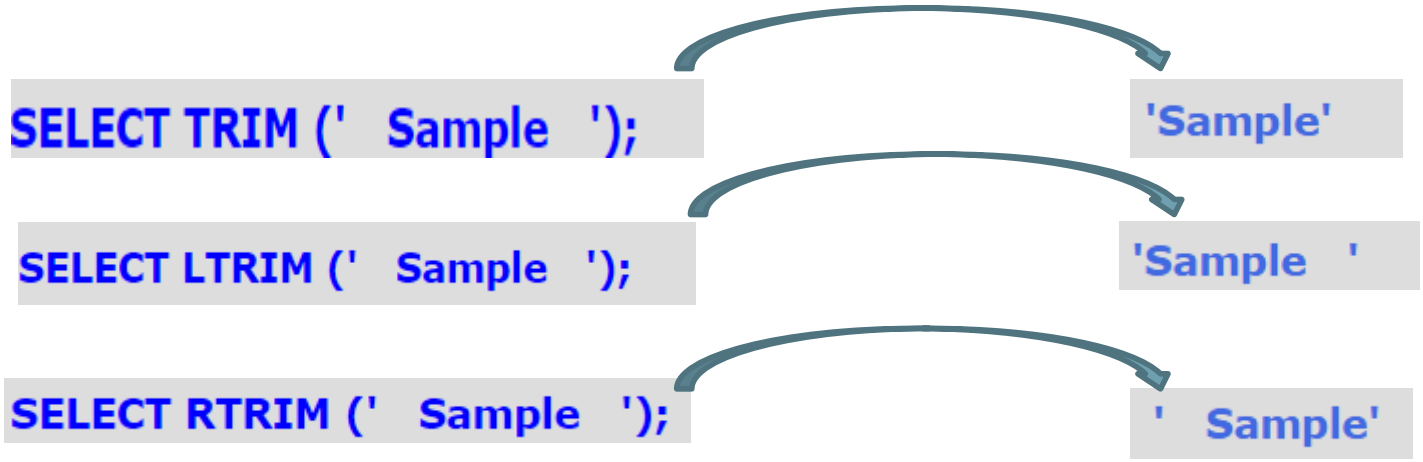
Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

SUBSTR(Store_Name,2,4)
an D

TRIM()

- Simple syntax: remove spaces

TRIM() LTRIM() RTRIM()



LENGTH()

- Simple syntax:

  Length(string)

```
SELECT Length (Store_Name)
FROM Geography
WHERE Store_Name = 'Los Angeles';
```

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

```
11
```

LENGTH()

- Simple syntax:

  Length(string)

```
SELECT Region_Name, Length (Region_Name)
FROM Geography;
```

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Region_Name	Length(Region_Name)
East	4
East	4
West	4
West	4

REPLACE()

- Simple syntax:

  Replace (column name, string2, string3)

```
SELECT REPLACE (Region_Name, 'ast', 'astern')
FROM Geography;
```

Geography

Region_Name	store_name
East	Boston
East	New York
West	Los Angeles
West	San Diego

REPLACE (Region_Name, 'ast', 'astern')
Eastern
Eastern
West
West

Varchar v.s. Char

Table Creation and Constraints

https://dev.mysql.com/doc/refman/8.0/en/dynindex-statement.html
https://www.1keydata.com/tw/sql/sql-length.html

CREATE TABLE

- Ex.

```
CREATE TABLE Customer
(First_Name char(50),
Last_Name char(50),
Address char(50),
City char(50),
Country char(25),
Birth_Date datetime);
```

SQL CONSTRAINT

```
CREATE TABLE Customer
(SID integer NOT NULL,
Last_Name varchar (30) NOT NULL,
First_Name varchar(30));
```

```
CREATE TABLE Customer
(SID integer UNIQUE,
Last_Name varchar (30),
First_Name varchar(30));
```

```
CREATE TABLE Customer
(SID integer CHECK (SID > 0),
Last_Name varchar (30),
First_Name varchar(30));
```

What happen?

Primary Key

```
CREATE TABLE Customer
(SID integer,
Last_Name varchar(30),
First_Name varchar(30),
PRIMARY KEY (SID));
```

```
ALTER TABLE Customer ADD PRIMARY KEY (SID);
```

Foreign Key

```
CREATE TABLE ORDERS
(Order_ID integer,
Order_Date date,
Customer_SID integer,
Amount double,
PRIMARY KEY (Order_ID),
FOREIGN KEY (Customer_SID) REFERENCES CUSTOMER (SID));
```

CREATE VIEW

Key differences between Table and View. A table is structured with columns and rows, while a view is a virtual table extracted from a database. The table is an independent data object while views are usually depending on the table. The table is an actual or real table that exists in physical locations.

```
CREATE VIEW V_Customer
AS SELECT First_Name, Last_Name, Country
FROM Customer;
```

CREATE INDEX

```
CREATE INDEX IDX_CUSTOMER_LAST_NAME
ON Customer (Last_Name);
```

ALTER TABLE

```
ALTER TABLE Customer ADD Gender char(1);
```

```
ALTER TABLE Customer CHANGE Address Addr char(50);
```

```
ALTER TABLE Customer MODIFY Addr char(30);
```

```
ALTER TABLE Customer DROP Gender;
```

DROP TABLE

```
DROP TABLE Customer;
```

INSERT INTO

```
INSERT INTO Store_Information (Store_Name, Sales, Txn_Date)
VALUES ('Los Angeles', 900, 'Jan-10-1999');
```

UPDATE

```
UPDATE Store_Information
SET Sales = 500
WHERE Store_Name = 'Los Angeles'
AND Txn_Date = 'Jan-08-1999';
```

DELETE

```
DELETE FROM Store_Information
WHERE Store_Name = 'Los Angeles';
```

Types of Databases

Types of Databases

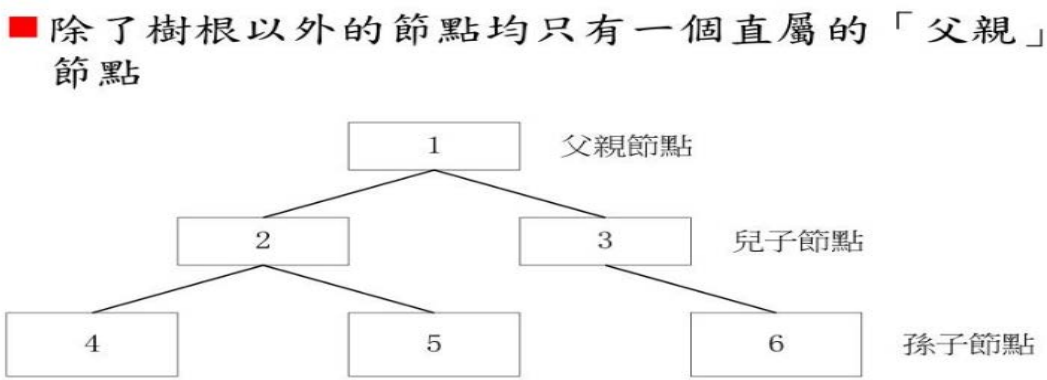
- A company should use the type of database that fit in its requirements and needs. There are various types of database structures:
  - **Relational databases-**Relational databases have been around since the 1970s. The name comes from the way that data is stored in multiple, related tables. Within the tables, data is stored in rows and columns. Organizations that have a lot of unstructured or semi-structured data should not be considering a relational database. (Examples of unstructured data are: Rich media. Media and entertainment data, surveillance data, geo-spatial data, audio, weather data. Semi Structured Data Examples: Email 、 CSV 、 XML and JSON documents 、 HTML, and so on)
    - Microsoft SQL Server, Oracle Database, MySQL, PostgreSQL and IBM DB2

Types of Databases

- A company should use the type of database that fit in its requirements and needs. There are various types of database structures: (We just introduce few types here!)
  - **NoSQL** is a broad category that includes any database that doesn't use SQL as its primary data access language.
    - These types of databases are also sometimes referred to as non-relational databases. Unlike in relational databases, data in a NoSQL database doesn't have to conform to a pre-defined schema, so these types of databases are great for organizations seeking to store unstructured or semi-structured data.
    - One advantage of NoSQL databases is that developers can make changes to the database on the fly, without affecting applications that are using the database.
    - Apache Cassandra, MongoDB, CouchDB, and CouchBase

Types of Databases

- A company should use the type of database that fit in its requirements and needs. There are various types of database structures: (We just introduce few types here!)
  - **Hierarchical databases** use a parent-child model to store data. If you were to draw a picture of a hierarchical database, it would look like a family tree, with one object on top branching down to multiple objects beneath it.
    - **Examples:** IBM Information Management System (IMS), Windows Registry



Types of Databases

- A company should use the type of database that fit in its requirements and needs. There are various types of database structures: (We just introduce few types here!)
  - **Document databases** Document databases, also known as document stores, use JSON-like documents to model data instead of rows and columns.
    - Sometimes referred to as document-oriented databases, document databases are designed to store and manage document-oriented information, also referred to as semi-structured data. Document databases are simple and scalable, making them useful for mobile apps that need fast iterations.
    - **Examples:** MongoDB, Amazon DocumentDB, Apache CouchDB

Why SQL Statements

What is a Relational Database (RDBMS)

- A relational database is a type of database that stores and provides access to data points that are related to one another.
- Relational databases are based on the relational model, an intuitive, straightforward way of representing data in tables.
- In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

Industry’s Best RDBMS

- Oracle database products offer customers cost-optimized and high-performance versions of Oracle Database, the world's leading converged, multi-model database management system, as well as in-memory, NoSQL and MySQL databases.
- Oracle Autonomous Database enables customers to simplify relational database environments and reduce management workloads.
- Good procedural computer language: PL/ SQL

Benefits of RDBMS

- The simple yet powerful relational model is used by organizations of all types and sizes for a broad variety of information needs.
- Relational databases are used to track inventories, process ecommerce transactions, manage huge amounts of mission-critical customer information, and much more.
- A relational database can be considered for any information need in which data points relate to each other and must be managed in a secure, rules-based, consistent way.
- Relational databases have been around since the 1970s. Today, the advantages of the relational model continue to make it the most widely accepted model for databases.

Relational model and data consistency

- The relational model is the best at maintaining data consistency across applications and database copies (called instances). For example, when a customer deposits money at an ATM and then looks at the account balance on a mobile phone, the customer expects to see that deposit reflected immediately in an updated account balance. Relational databases excel at this kind of data consistency, ensuring that multiple instances of a database have the same data all the time.
- It's difficult for other types of databases to maintain this level of **timely consistency** with large amounts of data. Some recent databases, such as NoSQL, can supply only "**eventual consistency**." Under this principle, when the database is scaled or when multiple users access the same data at the same time, the data needs some time to "catch up." Eventual consistency is acceptable for some uses, such as to maintain listings in a product catalog, but for critical business operations such as shopping cart transactions, the relational database is still the gold standard.

Q&A