Introduction to Databases and SQL

Software Testing

Agenda

- What is a database?
- Database types (RDBMS, NoSQL)
- Relational databases - basic terminology
- What is SQL?
- SQL queries

Databases

- **Database** is an organized collection of data
- **Database management systems** (**DBMS**s) are specially designed software applications that interact with the user, other applications, and the database itself to capture and analyze data
- Microsoft SQL Server, Oracle,MySQL, MariaDB, PostgreSQL, SQLite, SAP HANA, dBASE, FoxPro, IBM DB2, LibreOffice Base, FileMaker Pro and InterSystems Caché

Database types

- **Time Series Databases (TSDB)** - database optimized for time-stamped or time series data

- **NoSQL (Not Only SQL)** Databases

  Key-value stores -every item in the database is stored as an attribute name (or "key") together with its value

  Wide-column stores - store data together as columns instead of rows and are optimized for queries over large datasets

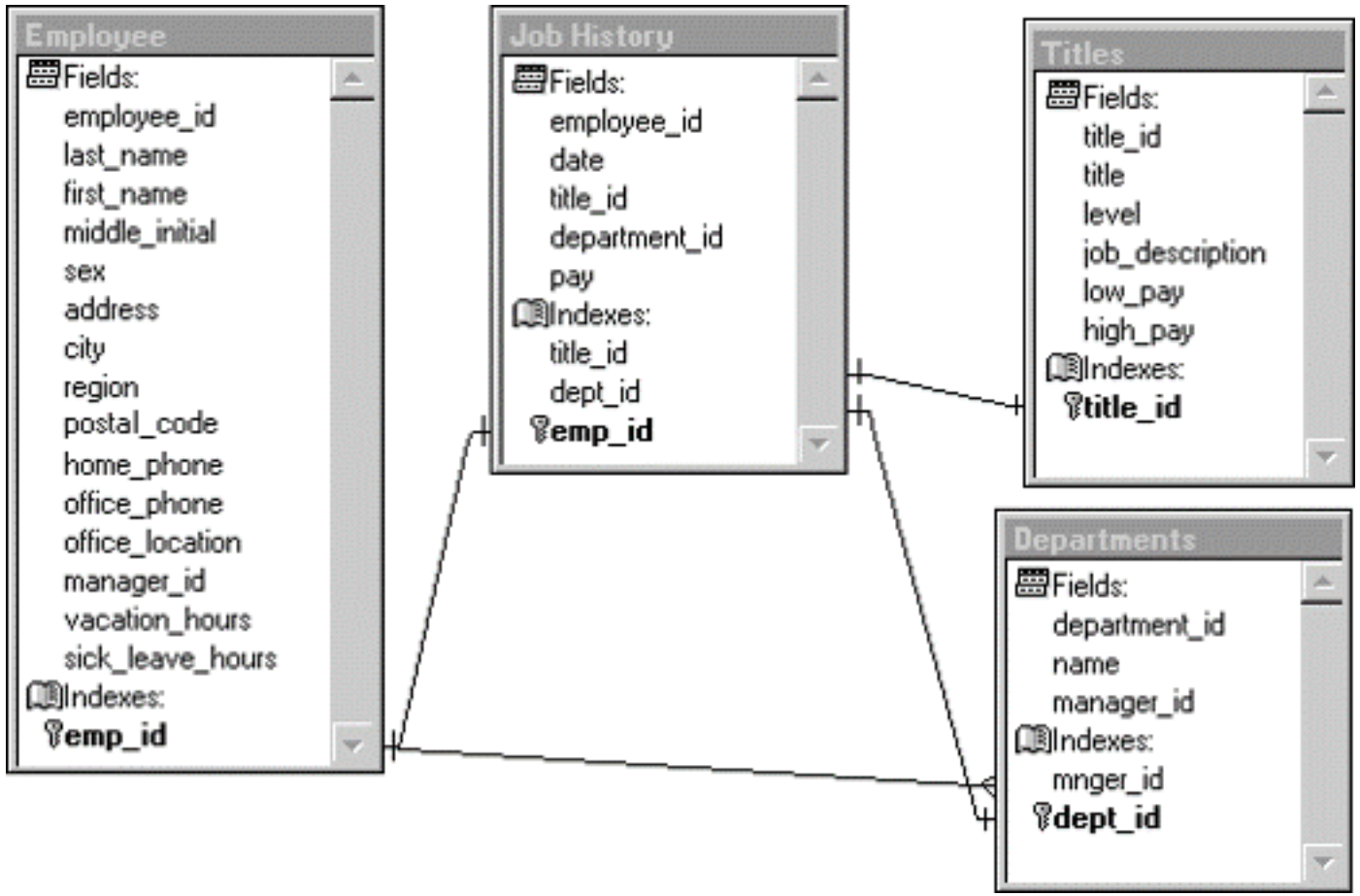
  Document databases - pair each key with a complex data structure known as a document

  Graph databases - used to store information about networks, such as social connections

- **Relational DBMS** - the database relationships are treated in the form of tables

SOFT ACADEMY

Relational DB model



Basic terminology

- **Table** - organized set of data elements (values) using a model of vertical columns (which are identified by their name) and horizontal rows. A table has a specified number of columns, but can have any number of rows

character_id	first_name	last_name	company_id
340234	Mickey	Mouse	1
456345	Bat	Man	3
345634	Donald	Duck	1
865474	Bugs	Bunny	4
245656	Iron	Man	2

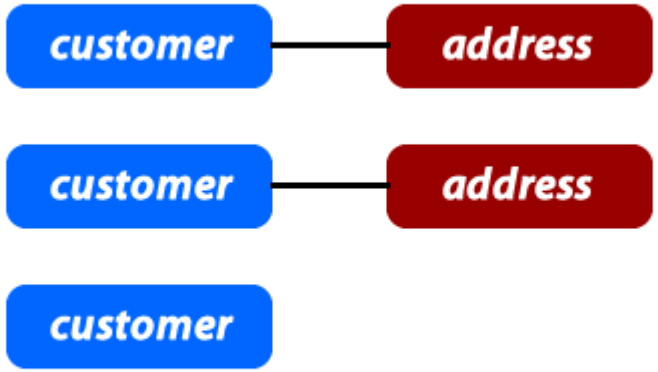
company_id	company_name
1	Disney
2	Marvel
3	DC Comics
4	Warner Bros

Basic terminology

- **Primary key** - uniquely identifies each record in a database table. Each table should have a primary key, and each table can have only ONE primary key. A primary key column cannot contain NULL values. Primary key can be built from more than one column.
- **Foreign key** - points to a PRIMARY KEY in another table. One table can have MORE than ONE foreign keys. Foreign key can contain NULL value.
- **NULL** value - indicate that a data value does not exist in the database

Basic terminology

Relationship types: One-to-One

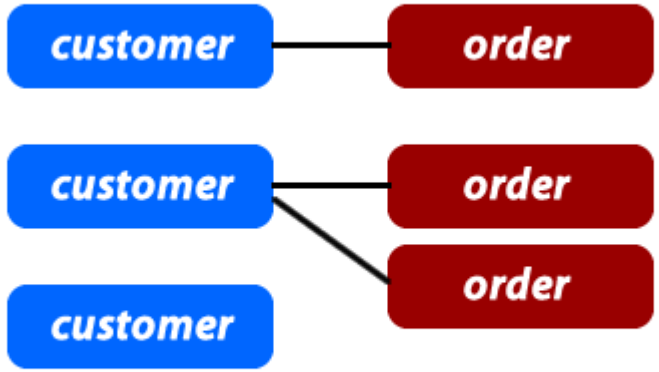


If each address can belong to only one customer, this relationship is "One to One".

Not very common. Typically this will be achieved with one table only.

Basic terminology

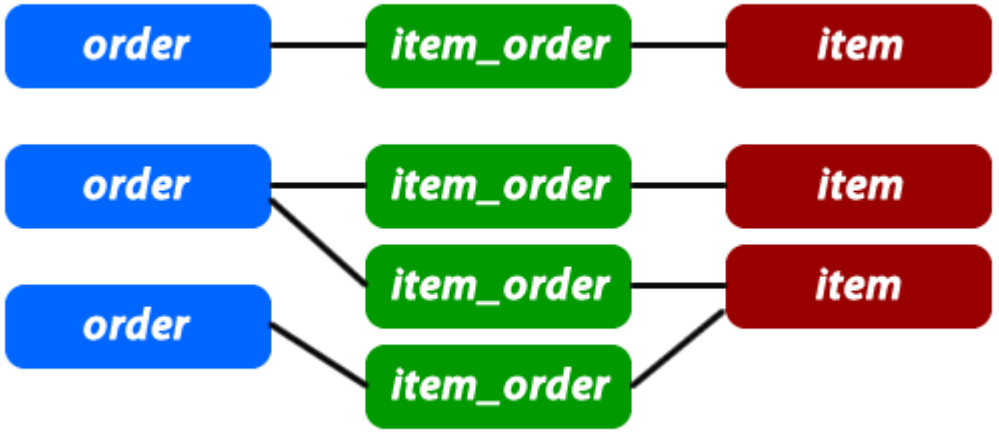
Relationship types: One-to-Many / Many-to-One



Most common case. Each customer may have zero, one or multiple orders. But an order can belong to only one customer.

Basic terminology

Relationship types: Many-to-Many



Each line of the junction table (in this case, ITEM_ORDER) connects one line from the left table (ORDER) with one line from the right table (ITEM).

Each primary key of ORDER can be copied many times to ITEM_ORDER; each primary key of ITEM can also be copied many times to ITEM_ORDER. But the same pair of Foreign keys in ITEM_ORDER can only occur once.

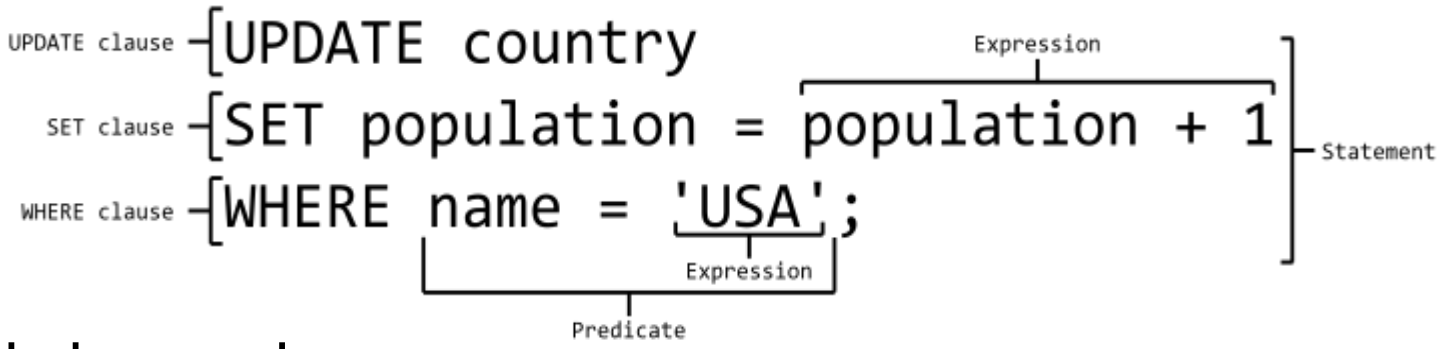
Data Types

- Each column in a database table is required to have a name and a data type.
- Most common data types:
  - **INT** – integer numbers
  - **DECIMAL** – decimal numbers
  - **CHAR** – fixed length string, up to 255 characters
  - **VARCHAR** – dynamic length string, up to 255 characters
  - **BOOLEAN** – true/false
  - **DATE** – date (YYYY-MM-DD)
  - **TIME** – time (HH-MM-SS)
  - **TIMESTAMP** – date and time (YYYY-MM-DD HH-MM-SS)

What is SQL?

- SQL stands for Structured Query Language
- Language for manipulating data in relational databases (DB)
- Minor differences in implementations between SQL vendors

SQL Structure



- **Clauses** - components of statement
- **Expression** - produces values or tables
- **Queries** - which retrieve the data based on specific criteria
- **Statements** - which may have a persistent effect on data

Data definition language (DDL)

- CREATE
- DROP
- ALTER
- TRUNCATE

Create

```
CREATE TABLE table_name
(
column_name1 data_type(size),
column_name2 data_type(size),
column_name3 data_type(size),
....
);

CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

DROP and TRUNCATE

```
DROP TABLE table_name;

DROP DATABASE database_name;

DROP TABLE persons;

DROP DATABASE stores;
```

Truncate is used only with TABLES

```
TRUNCATE TABLE table_name;
TRUNCATE TABLE persons;
```

ALTER

```
ALTER TABLE table_name
ADD column_name datatype;

ALTER TABLE table_name
DROP COLUMN column_name;

ALTER TABLE Persons
ADD DateOfBirth date;

ALTER TABLE Persons
DROP COLUMN DateOfBirth;
```

Data manipulation language (DML)

- SELECT
- INSERT
- UPDATE
- DELETE

SELECT

The SELECT statement is used to select data from a database.

SELECT *column_name*,*column_name*
FROM *table_name*;

SELECT * FROM *table_name*;

SELECT *firstname*, *lastname*
FROM *persons*;

SELECT * FROM *persons*;

SELECT DISTINCT

The SELECT DISTINCT statement is used to return only distinct (different) values.

SELECT DISTINCT *column_name*,*column_name*
FROM *table_name*;

SELECT DISTINCT City FROM Customers;

SELECT DISTINCT *firstname*, *lastname*
FROM *persons*;

WHERE clause

The WHERE clause is used to filter records.

```
SELECT column_name,column_name
FROM table_name
WHERE column_name operator value;
```

```
SELECT *
FROM table_name
WHERE column_name operator value;
```

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

```
SELECT Name
FROM persons
WHERE age > =18;
```

AND/or operators

The AND operator displays a record if both the first condition AND the second condition are true.

The OR operator displays a record if either the first condition OR the second condition is true.

```
SELECT * FROM Customers
WHERE Country='Germany'
AND City='Berlin';

SELECT * FROM Customers
WHERE City='Berlin'
OR City='München';

SELECT * FROM Customers
WHERE Country='Germany'
AND (City='Berlin' OR City='München');
```

ORDER BY

The ORDER BY keyword is used to sort the result-set.

SELECT *column_name,column_name*
FROM *table_name*
ORDER BY *column_name,column_name* ASC|DESC;

```
SELECT * FROM Customers
ORDER BY Country;
```

```
SELECT * FROM Customers
ORDER BY Country DESC;
```

```
SELECT * FROM Customers
ORDER BY Country,CustomerName;
```

LIKE operator and wildcards

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;

SELECT * FROM Customers
WHERE City LIKE 's%';

SELECT * FROM Customers
WHERE City LIKE '%s';

SELECT * FROM Customers
WHERE Country LIKE '%ari%';

SELECT * FROM Customers
WHERE Country NOT LIKE '%ari%';
```

IN operator

The IN operator allows you to specify multiple values in a WHERE clause.

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*value1,value2,...*);

SELECT * FROM Customers
WHERE City IN ('Paris','London');

BETWEEN operator

The IN operator allows you to specify multiple values in a WHERE clause.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

```
SELECT * FROM Products
WHERE Price BETWEEN 10 AND 20;
```

```
SELECT * FROM Products
WHERE Price NOT BETWEEN 10 AND 20;
```

```
SELECT * FROM Products
WHERE (Price BETWEEN 10 AND 20)
AND NOT CategoryID IN (1,2,3);
```

```
SELECT * FROM Products
WHERE ProductName BETWEEN 'C' AND 'M';
```

INSERT INTO statement

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO *table_name*
VALUES (*value1,value2,value3,...*);

INSERT INTO *table_name* (*column1,column2,column3,...*)
VALUES (*value1,value2,value3,...*);

INSERT INTO *table2*
SELECT * FROM *table1*;

INSERT INTO *table2*
(column_name(s))
SELECT *column_name(s)*
FROM *table1*;

INSERT INTO statement

```
INSERT INTO Customers (CustomerName, ContactName, Address, City,
PostalCode, Country)
VALUES ('Cardinal','Tom B. Erichsen','Skagen 21','Stavanger','4006','Norway');

INSERT INTO Customers (CustomerName, City, Country)
VALUES ('Cardinal', 'Stavanger', 'Norway');

INSERT INTO Customers (CustomerName, Country)
SELECT SupplierName, Country FROM Suppliers;

INSERT INTO Customers (CustomerName, Country)
SELECT SupplierName, Country FROM Suppliers
WHERE Country='Germany';
```

UPDATE statement

The UPDATE statement is used to update existing records in a table.

UPDATE *table_name*
SET *column1=value1,column2=value2,...*
WHERE *some_column=some_value*;

```
UPDATE Customers
SET ContactName='Alfred Schmidt', City='Hamburg'
WHERE CustomerName='Alfreds Futterkiste';
```

DELETE statement

The DELETE statement is used to delete records in a table.

DELETE FROM *table_name*
WHERE *some_column=some_value*;

```
DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria Anders';
```

JOIN clauses

An SQL JOIN clause is used to combine rows from two or more tables, based on a common field between them.

JOIN types:

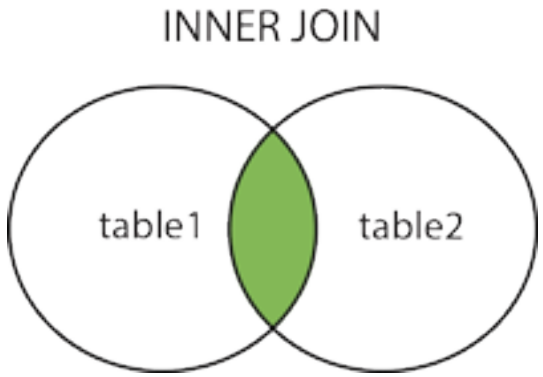
INNER JOIN - return all rows from multiple tables where the join condition is met

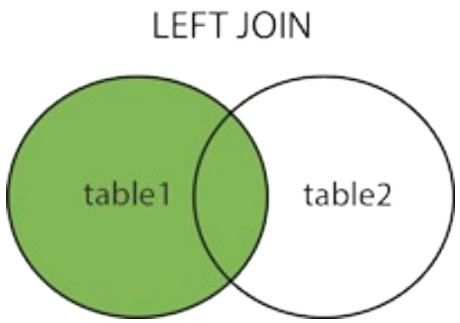
LEFT JOIN keyword returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is NULL in the right side when there is no match.

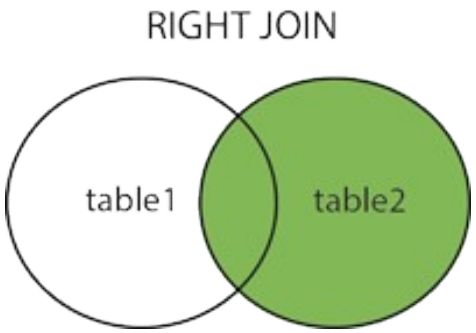
RIGHT JOIN - The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match.

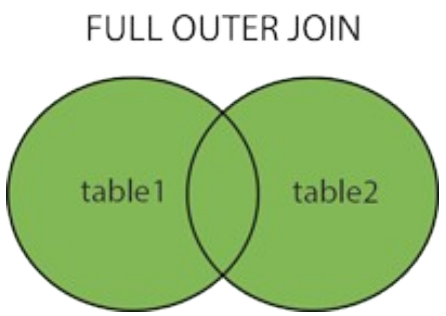
FULL JOIN - The FULL OUTER JOIN keyword returns all rows from the left table (table1) and from the right table (table2).

JOIN clauses









JOIN clauses

JOIN Syntax:

SELECT *column_name(s)*
FROM *table1*
(INNER/LEFT/RIGHT/FULL OUTER) JOIN *table2*
ON *table1.column_name=table2.column_name*;

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
INNER JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders
ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

GROUP BY and Aggregate functions

The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

SELECT *column_name, aggregate_function(column_name)*
FROM *table_name*
WHERE *column_name* operator *value*
GROUP BY *column_name*;

The following SQL statement counts as orders grouped by shippers:

```
SELECT Shippers.ShipperName,COUNT(Orders.OrderID) AS NumberOfOrders
FROM Orders
LEFT JOIN Shippers
ON Orders.ShipperID=Shippers.ShipperID
GROUP BY ShipperName;
```

GROUP BY and Aggregate functions

Useful Aggregate functions:

AVG()
COUNT()
SUM()
MAX()
MIN()

Resources

- SQL tutorial
  http://www.w3schools.com/sql/

- SQLiteBrowser
  https://sqlitebrowser.org/

SOFT ACADEMY