Web Programming

YJ – 2016

MySQL and AJAX

- ❖ Database Concepts
- ❖ SQL
- ❖ ER Diagram
- ❖ Case Study
- ❖ Ajax

SQL

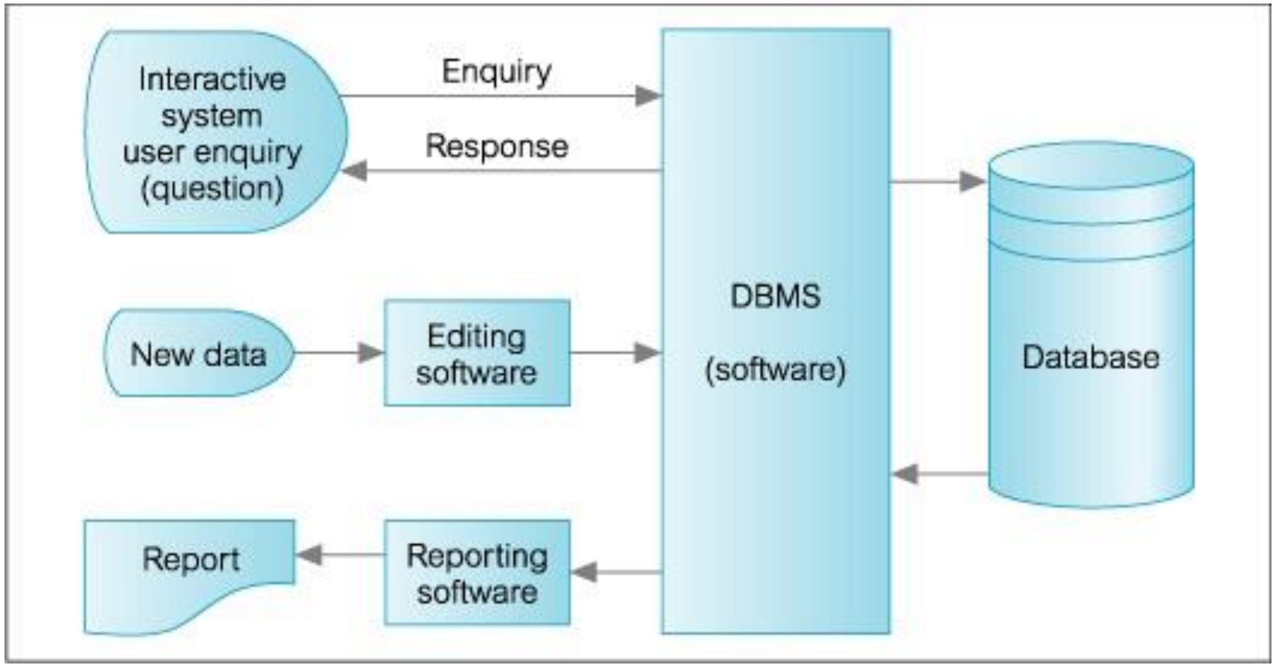
SQL is a database computer language designed for the retrieval and management of data in relational database. SQL stands for Structured Query Language.

SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as standard database language.

- ❖ Allows users to access data in relational database management systems.
- ❖ Allows users to describe the data.
- ❖ Allows users to define the data in database and manipulate that data.
- ❖ Allows to embed within other languages using SQL modules, libraries & pre-compilers.
- ❖ Allows users to create and drop databases and tables.
- ❖ Allows users to create view, stored procedure, functions in a database.
- ❖ Allows users to set permissions on tables, procedures, and views

RDBMS

- RDBMS stands for **R**elational **D**atabase **M**anagement **S**ystem. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.



Popular RDBMS

- MySQL

  MySQL is an open source SQL database, which is developed by Swedish company MySQL AB. MySQL comes with a very fast, multi-threaded, multi-user, and robust SQL database server.

- MS SQL Server

  MS SQL Server is a Relational Database Management System developed by Microsoft Inc.

- ORACLE

  It is a very large and multi-user database management system. Oracle is a relational database management system developed by 'Oracle Corporation'.

  Oracle works to efficiently manage its resource, a database of information, among the multiple clients requesting and sending data in the network.

  It is an excellent database server choice for client/server computing. Oracle supports all major operating systems for both clients and servers, including MSDOS, NetWare, UnixWare, OS/2 and most UNIX flavors.

- MS ACCESS

  Microsoft Access is an entry-level database management software. MS Access database is not only an inexpensive but also powerful database for small-scale projects.

SQL Syntax

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name
HAVING (ARITHMETIC FUNCTION CONDITION)
ORDER BY COLUMN1, COLUMN2

UNION

SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name
HAVING (arithmetic function condition)
ORDER BY COLUMN1, COLUMN2
;
```

Some Concepts

- Table
- Field
- Row
- Column
- Constraints
  - NOT NULL Constraint: Ensures that a column cannot have NULL value.
  - DEFAULT Constraint: Provides a default value for a column when none is specified.
  - UNIQUE Constraint: Ensures that all values in a column are different.
  - PRIMARY Key: Uniquely identified each rows/records in a database table.
  - FOREIGN Key: Uniquely identified a rows/records in any another database table.
  - CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.
  - INDEX: Use to create and retrieve data from the database very quickly.

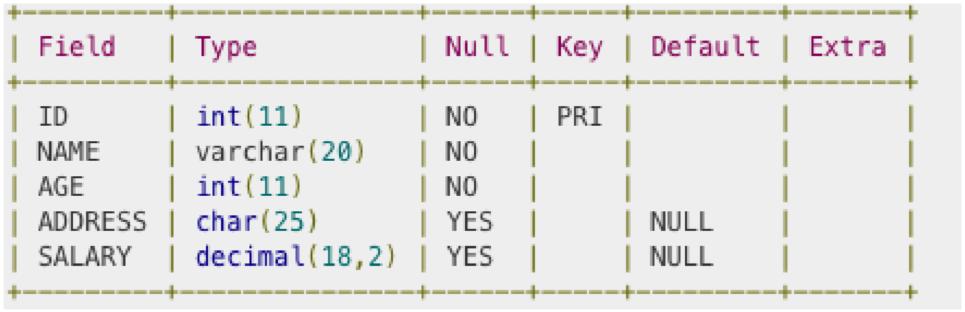
SQL Syntax

```
SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name
HAVING (ARITHMETIC FUNCTION CONDITION)
ORDER BY COLUMN1, COLUMN2

UNION

SELECT SUM(column_name)
FROM   table_name
WHERE  CONDITION
GROUP BY column_name
HAVING (arithmetic function condition)
ORDER BY COLUMN1, COLUMN2
;
```

Command

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

```
CREATE DATABASE DatabaseName;

CREATE TABLE CUSTOMERS(
  ID   INT            NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
  AGE  INT            NOT NULL,
  ADDRESS  CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```



```
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| ID      | int(11)       | NO   | PRI |         |       |
| NAME    | varchar(20)   | NO   |     |         |       |
| AGE     | int(11)       | NO   |     |         |       |
| ADDRESS | char(25)      | YES  |     | NULL    |       |
| SALARY  | decimal(18,2) | YES  |     | NULL    |       |
+---------+---------------+------+-----+---------+-------+
```

Command

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

```
DROP DATABASE DatabaseName;

DROP TABLE table_name;

ALTER TABLE table_name ADD column_name datatype;

ALTER TABLE table_name DROP COLUMN column_name;

ALTER TABLE table_name MODIFY COLUMN column_name datatype;
```

Command

Command	Description
INSERT	Creates a record
SELECT	Retrieves certain records from one or more tables
UPDATE	Modifies records
DELETE	Deletes records

Insert

```
INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);

INSERT INTO TABLE_NAME VALUES (value1,value2,value3,...valueN);

INSERT INTO `City` (`ID`, `Name`, `CountryCode`, `District`, `Population`)
VALUES
  (1893, 'Tianjin', 'CHN', 'Tianjin', 5286800);
```

Update & Delete

```
UPDATE table_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];

UPDATE CITY SET POPULATION = '6286800' WHERE ID='1893';
```

```
DELETE FROM table_name WHERE [condition];

DELETE FROM CITY WHERE ID='1893';
```

Select, Where, AND&OR

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] AND [condition2]...AND [conditionN];
```

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition1] OR [condition2]...OR [conditionN];
```

```
SELECT * FROM CITY WHERE COUNTRYCODE = 'CHN’;
```

```
SELECT Population FROM CITY WHERE Name= ’Tianjin';
```

```
SELECT * FROM CITY WHERE COUNTRYCODE = 'CHN' AND NAME='TIANJIN';
```

```
SELECT * FROM CITY WHERE NAME='TIANJIN' OR NAME='WUHAN';
```

Select … As

```
SELECT column1, column2....
FROM table_name AS alias_name
WHERE [condition];
```

```
SELECT column_name AS alias_name
FROM table_name
WHERE [condition];
```

```
SELECT C.ID, C.NAME, C.AGE, O.AMOUNT
       FROM CUSTOMERS AS C, ORDERS AS O
       WHERE  C.ID = O.CUSTOMER_ID;
```

Distinct

The SQL **DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

```
SELECT DISTINCT column1, column2,.....columnN
FROM table_name
WHERE [condition];
```

```
SELECT DISTINCT COUNTRYCODE FROM CITY;
```

Like

Statement	Description
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and are at least 3 characters in length
WHERE SALARY LIKE '%2'	Finds any values that end with 2
WHERE SALARY LIKE '_2%3'	Finds any values that have a 2 in the second position and end with a 3
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

```sql
SELECT * FROM CITY WHERE NAME like 'TIANJI_’;
```

Order By

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

```
SELECT column_list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

```
SELECT NAME FROM CITY WHERE COUNTRYCODE = 'CHN' ORDER BY NAME;
```

Group By

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2;
```

```
SELECT COUNT(*), CONTINENT FROM COUNTRY GROUP BY CONTINENT;
```

Having

The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

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

```
SELECT COUNT(*), CONTINENT
FROM COUNTRY
GROUP BY CONTINENT
HAVING COUNT(*) > 30;
```

Limit

The LIMIT clause is used to specify the number of records to return.
The LIMIT clause can be very useful on large tables with thousands of records.
Returning a large number of records can impact on performance.

```
SELECT column_name FROM table_name
WHERE [condition]
LIMIT 2
ORDER BY column_name DESC;

SELECT * FROM CITY LIMIT 10 OFFSET 15;
```

Functions

- SQL COUNT Function

  The SQL COUNT aggregate function is used to count the number of rows in a database table.

  SELECT *COUNT*(*) **FROM** COUNTRY **WHERE** CONTINENT=**'ASIA'**;

- SQL MAX Function

  The SQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.

  SELECT NAME, POPULATION **FROM** CITY

  WHERE POPULATION = (**SELECT** *MAX*(POPULATION) **FROM** CITY);

- SQL MIN Function

  The SQL MIN aggregate function allows us to select the lowest (minimum) value for a certain column.

  SELECT NAME, POPULATION **FROM** CITY

  WHERE POPULATION = (**SELECT** *MIN*(POPULATION) **FROM** CITY);

- SQL AVG Function

  The SQL AVG aggregate function selects the average value for certain table column.

  SELECT *AVG*(POPULATION), COUNTRYCODE **FROM** CITY **GROUP BY** COUNTRYCODE;

Functions

❖ SQL SUM Function

The SQL SUM aggregate function allows selecting the total for a numeric column.

```
SELECT SUM(POPULATION), COUNTRYCODE FROM CITY
GROUP BY COUNTRYCODE
ORDER BY SUM(POPULATION) DESC;
```

❖ SQL SQRT Functions

This is used to generate a square root of a given number.

```
SELECT SQRT(16);
```

❖ SQL RAND Function

This is used to generate a random number using SQL command.

```
SELECT RAND( );

ORDER BY RAND();
```

Join

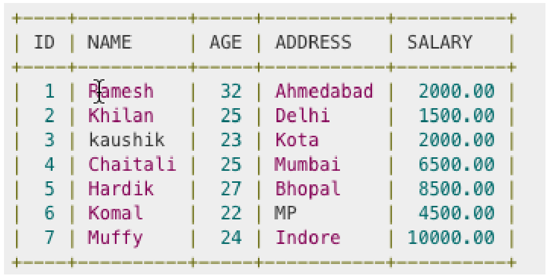
The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

The Joining Process

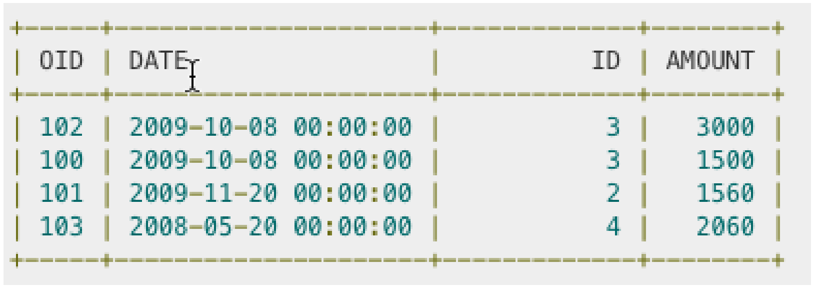
1. Combine every tuple in the first relation with every tuple in all other relations in the FROM clause.
2. Apply the joining condition from the WHERE clause.
3. Project onto the list of attributes and expressions in the SELECT clause.

- ❖ INNER JOIN: Returns all rows when there is at least one match in BOTH tables
- ❖ LEFT JOIN: Return all rows from the left table, and the matched rows from the right table
- ❖ RIGHT JOIN: Return all rows from the right table, and the matched rows from the left table
- ❖ FULL JOIN: Return all rows when there is a match in ONE of the tables

Inner Join

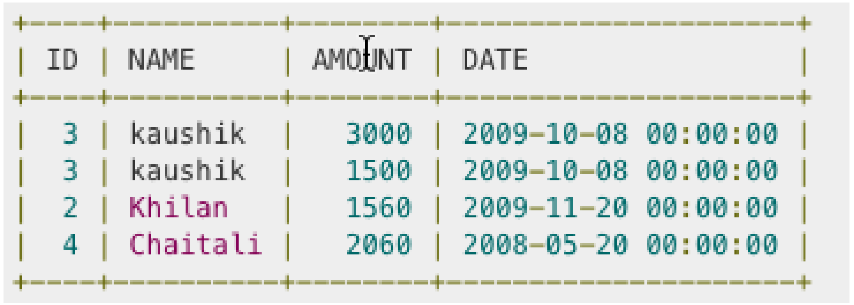


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

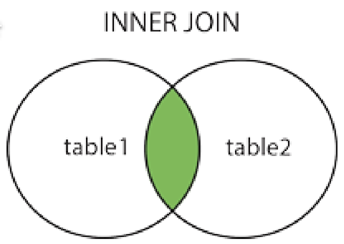


```
+-----+---------------------+-------------+--------+
| OID | DATE                |          ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

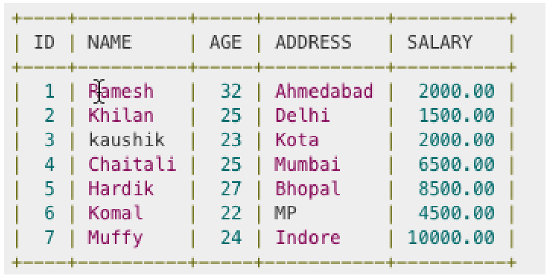
```
SELECT  ID, NAME, AMOUNT, DATE
    FROM CUSTOMERS
    INNER JOIN ORDERS
    ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```



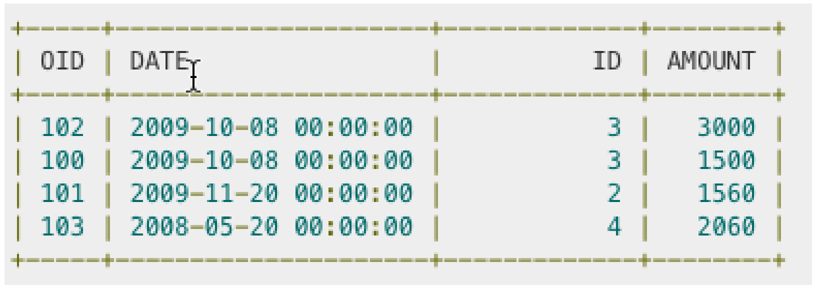
```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+----+----------+--------+---------------------+
```



Left Join

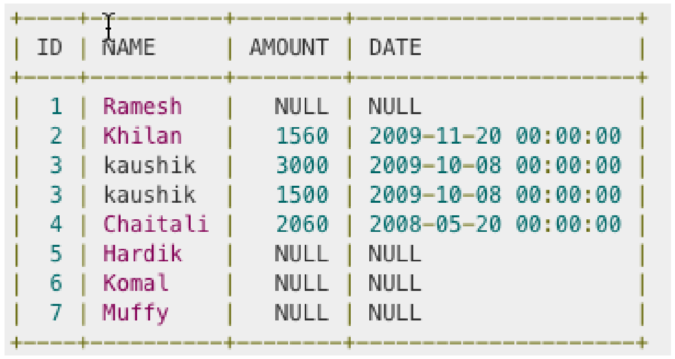


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

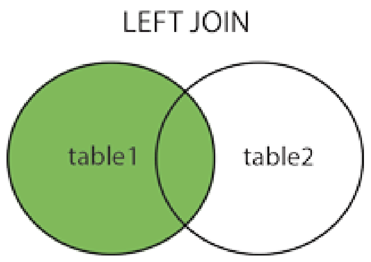


```
+-----+---------------------+-------------+--------+
| OID | DATE                |          ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

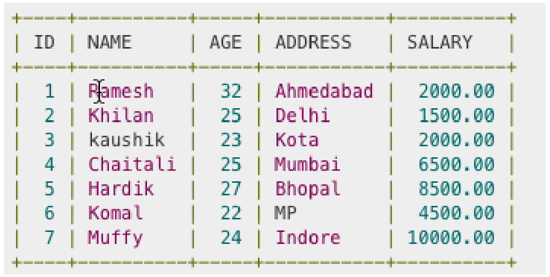
```
SELECT  ID, NAME, AMOUNT, DATE
    FROM CUSTOMERS
    LEFT JOIN ORDERS
    ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```



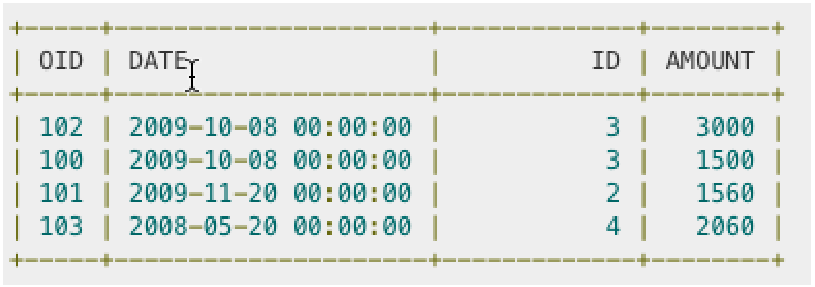
```
+----+----------+--------+---------------------+
| ID | NAME     | AMOUNT | DATE                |
+----+----------+--------+---------------------+
|  1 | Ramesh   |   NULL | NULL                |
|  2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|  3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|  3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|  4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|  5 | Hardik   |   NULL | NULL                |
|  6 | Komal    |   NULL | NULL                |
|  7 | Muffy    |   NULL | NULL                |
+----+----------+--------+---------------------+
```



Right Join

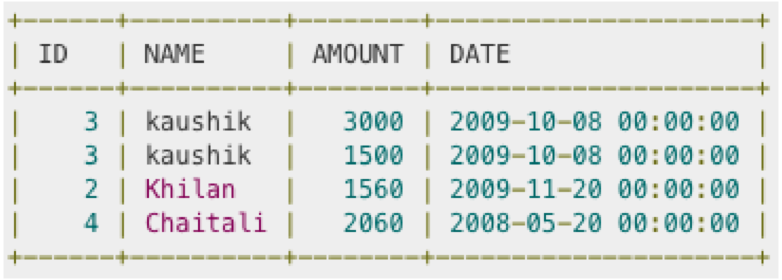


```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

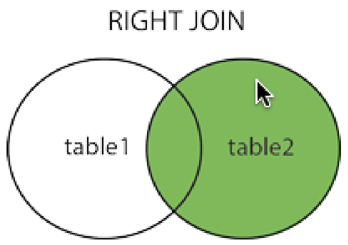


```
+-----+---------------------+-------------+--------+
| OID | DATE                |          ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

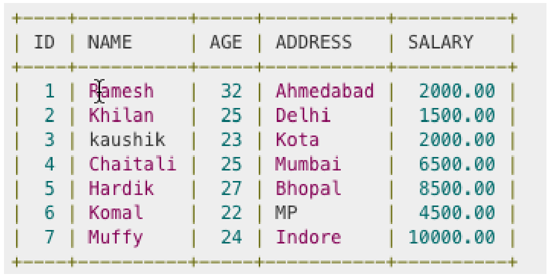
```
SELECT  ID, NAME, AMOUNT, DATE
    FROM CUSTOMERS
    RIGHT JOIN ORDERS
    ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```



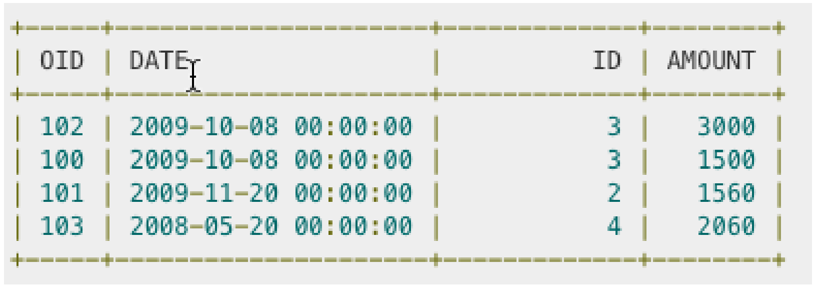
```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```



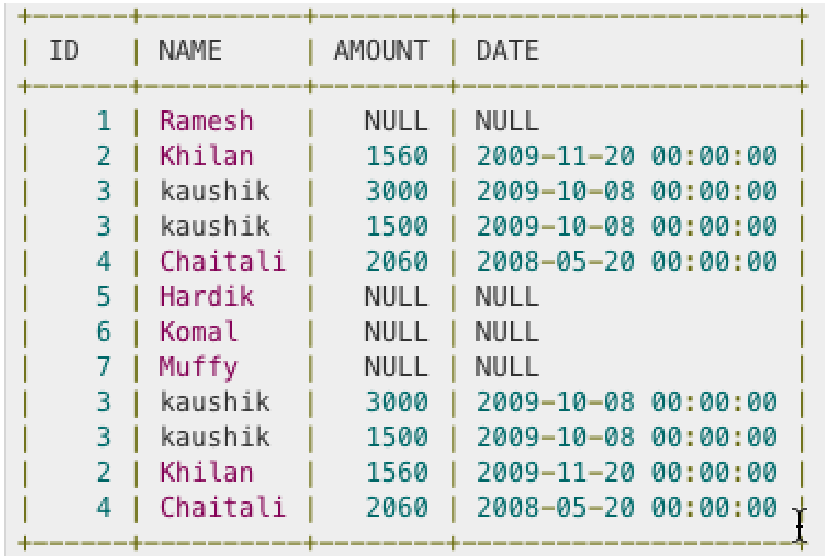
Full Join / Union All



```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```



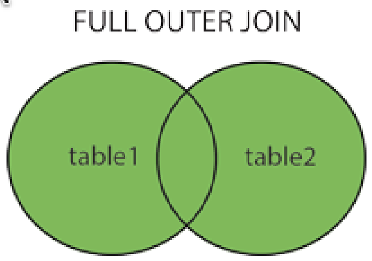
```
+-----+---------------------+-------------+--------+
| OID | DATE                |          ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```



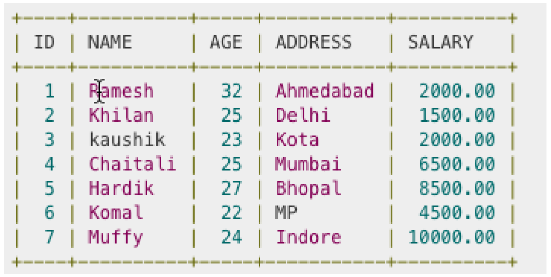
```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
+------+----------+--------+---------------------+
```

```
SELECT  ID, NAME, AMOUNT, DATE
FROM CUSTOMERS
LEFT JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
   UNION ALL
SELECT  ID, NAME, AMOUNT, DATE
FROM CUSTOMERS
RIGHT JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

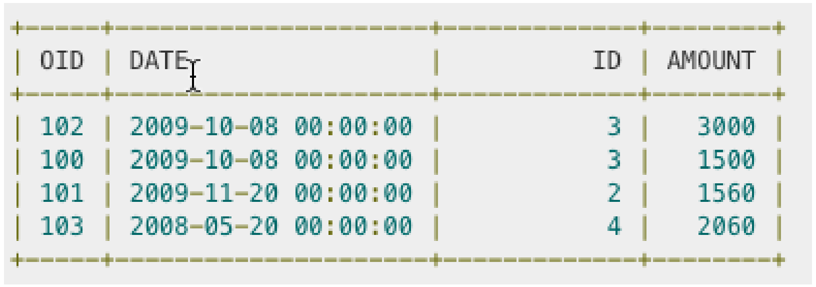
```
SELECT  ID, NAME, AMOUNT, DATE
FROM CUSTOMERS
FULL JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```



Union



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

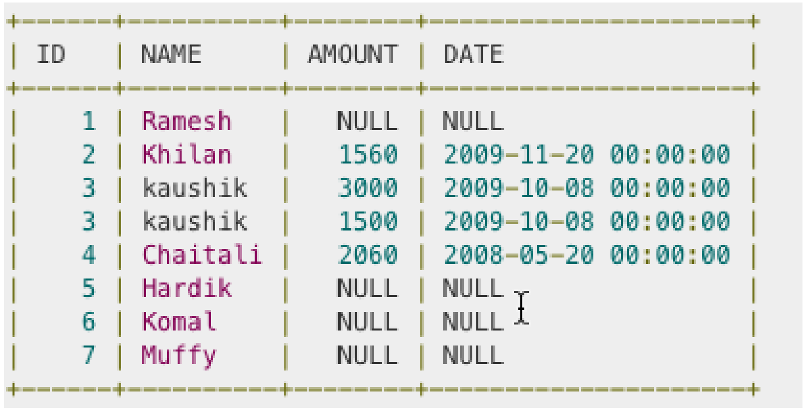


OID	DATE	ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

SELECT ID, NAME, AMOUNT, ***DATE***
FROM CUSTOMERS
LEFT JOIN ORDERS
ON CUSTOMERS.**ID** = ORDERS.CUSTOMER_ID

UNION

SELECT ID, NAME, AMOUNT, ***DATE***
FROM CUSTOMERS
RIGHT JOIN ORDERS
ON CUSTOMERS.**ID** = ORDERS.CUSTOMER_ID;

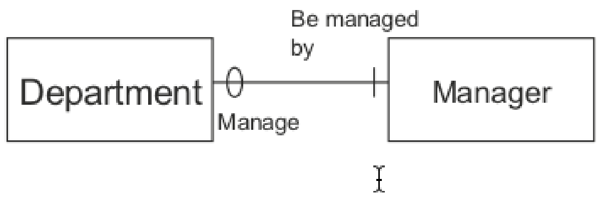


ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

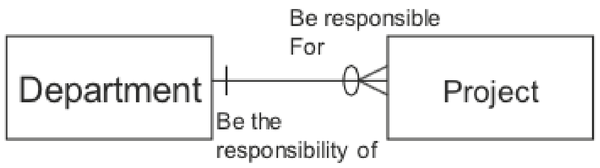
Entity Relationship

- Database is more than data, it has the relationship between data tables (Entities).
- An entity–relationship model is the result of using a systematic process to describe and define a subject area of business data. It is a graphical approach to representing the structure of information. Sometimes known as a data structure diagram (DSD, ERD etc.)
- Actually quite simple, with two basic elements:
  - Entities
  - Relationships
- Models the things in the real world that the information system needs to represent, and the specific items of information about those things.

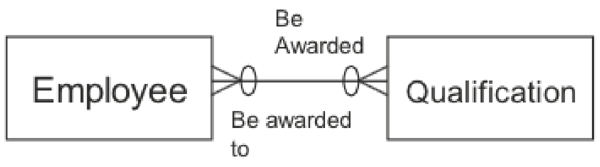
Entity Relationship



One-To-One

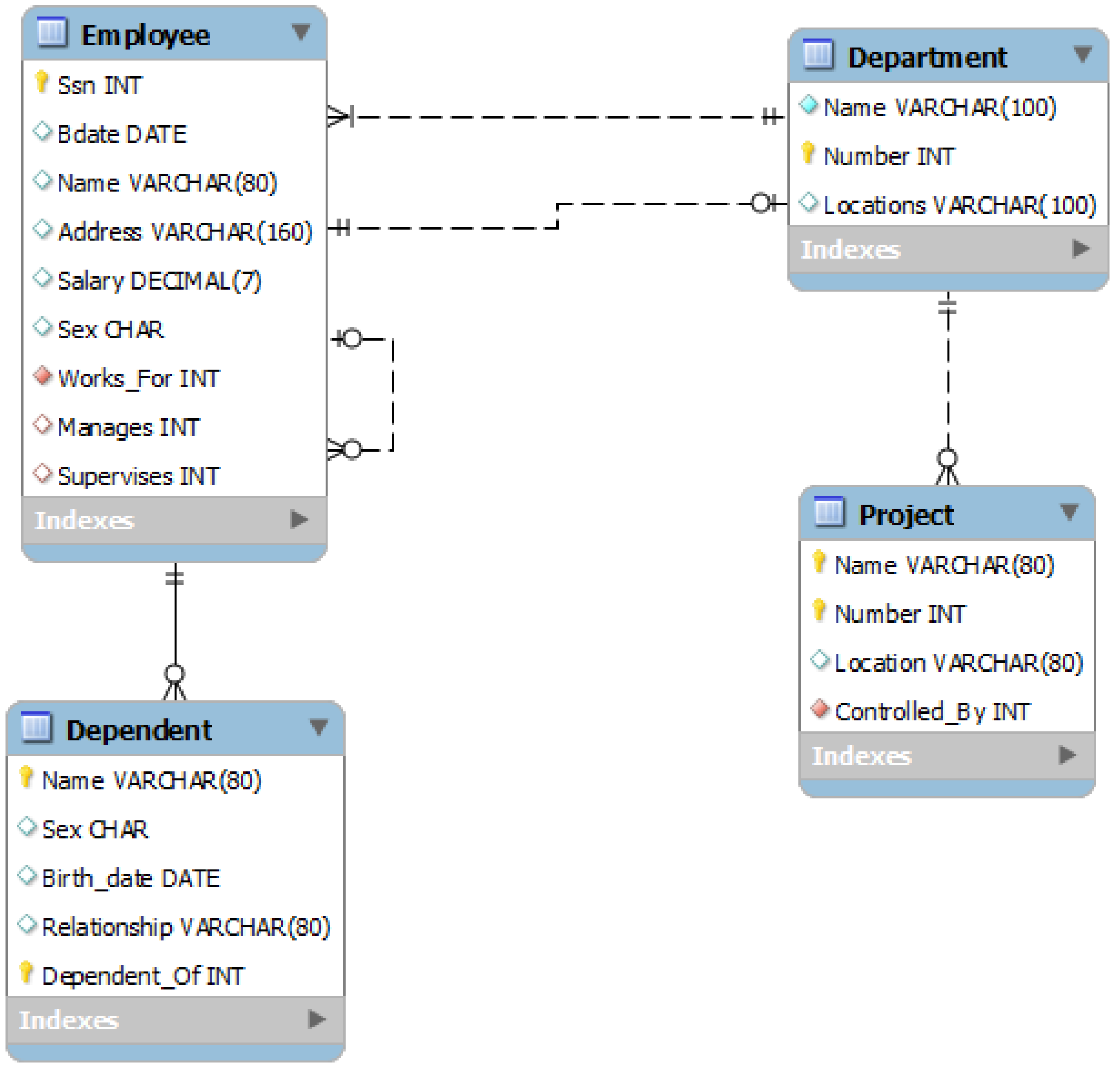


One-To-Many



Many-To-Many

Entity Relationship



Functional Dependency

An attribute B is FUNCTIONALLY DEPENDENT on another attribute A, if a value of A determines a single value of B at any one time.

ORDER-NUMBER➔ORDER-DATE

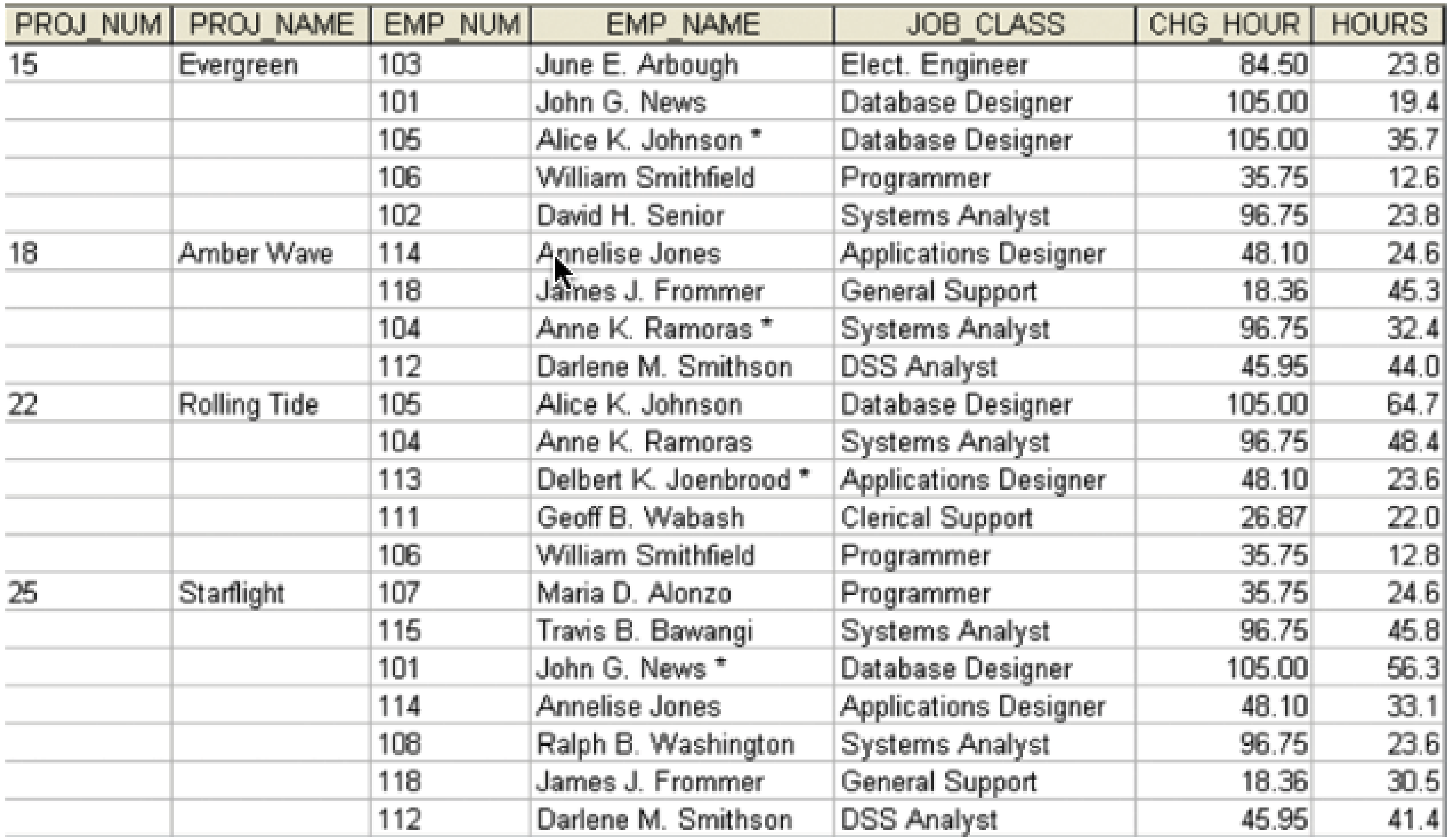
ORDER-NUMBER, PART-NUMBER ➔ QTY-ORDERED, PART-DESCRIPTION

– here although qty-ordered is **fully dependent** on order-number and part-number, only part-number is required to determine part-description
– part-description is said to be **partially dependent** on order-number and part- number

INVOICE-NUMB ➔ CUSTOMER-NUMB ➔ CUSTOMER-NAME

- **transitive dependency** occurs when Y depends on X, and Z depends on Y - thus Z also depends on X ie. X➔Y➔Z

Unormalised Form (UNF)



PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
		101	John G. News	Database Designer	105.00	19.4
		105	Alice K. Johnson *	Database Designer	105.00	35.7
		106	William Smithfield	Programmer	35.75	12.6
		102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
		118	James J. Frommer	General Support	18.36	45.3
		104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
		112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
		104	Anne K. Ramoras	Systems Analyst	96.75	48.4
		113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
		111	Geoff B. Wabash	Clerical Support	26.87	22.0
		106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
		115	Travis B. Bawangi	Systems Analyst	96.75	45.8
		101	John G. News *	Database Designer	105.00	56.3
		114	Annelise Jones	Applications Designer	48.10	33.1
		108	Ralph B. Washington	Systems Analyst	96.75	23.6
		118	James J. Frommer	General Support	18.36	30.5
		112	Darlene M. Smithson	DSS Analyst	45.95	41.4

First Normal Form (1NF)

A RELATION IS IN FIRST NORMAL FORM (1NF) IF

- ❖ a unique key has been identified for each tuple/row.
- ❖ it is a valid relation
  - > – Entity integrity (no part of PK is null)
  - > – Single value for each cell.
  - > – No repeating group.
- ❖ all attributes are functionally dependent on all or part of the primary key

2NF & 3NF

A RELATION IS IN 2NF IF -

- all non key attributes are functionally dependent on the entire key
- ie. no partial dependencies exist

A RELATION IS IN 3NF IF –

- all transitive dependencies have been removed - check for non key attribute dependent on another non key attribute
- Move from 2NF to 3NF by removing transitive dependencies

Entire Process UNF to 3NF

- UNF
  PROJECT (proj_num, proj_name {emp_num, emp_name, job_class, chg_hour, assign_hours})

- 1NF – remove repeating group
  PROJECT (proj_num, proj_name)
  ASSIGN (proj_num, emp_num, emp_name, job_class, chg_hour, assign_hours)

- 2NF – remove partial dependencies
  PROJECT (proj_num, proj_name)
  EMPLOYEE (emp_num, emp_name, job_class, chg_hour)
  ASSIGN (proj_num, emp_num, assign_hours)

- 3NF
  PROJECT (proj_num, proj_name)
  EMPLOYEE (emp_num, emp_name, job_class)
  ASSIGN (proj_num, emp_num, assign_hours)
  JOB (job_class, chg_hour)

Case Study

Pizza shop

Types of pizza

Orders

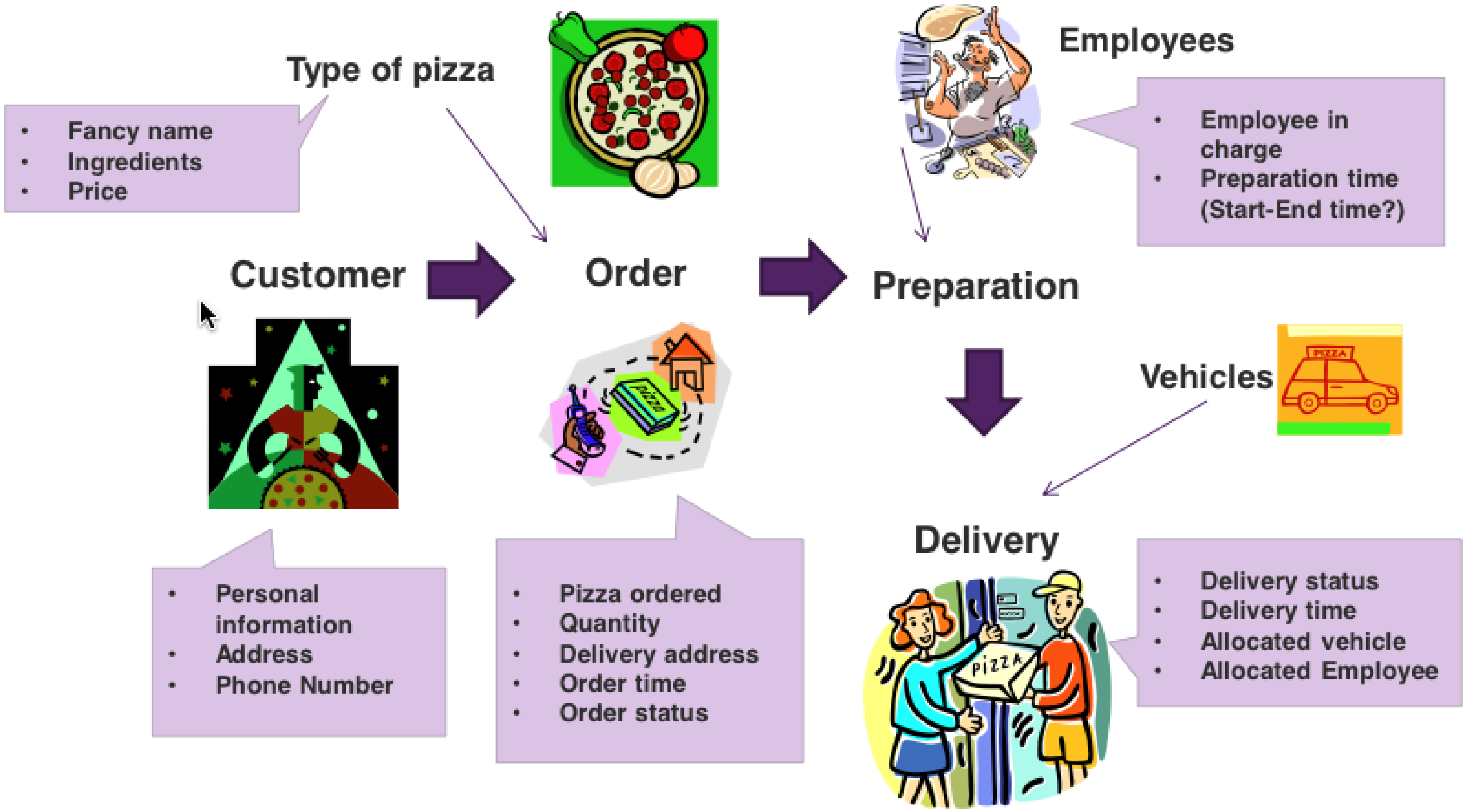
Delivery

Employees

Vehicles

Customers

Case Study



- Fancy name
- Ingredients
- Price

- Employee in charge
- Preparation time (Start-End time?)

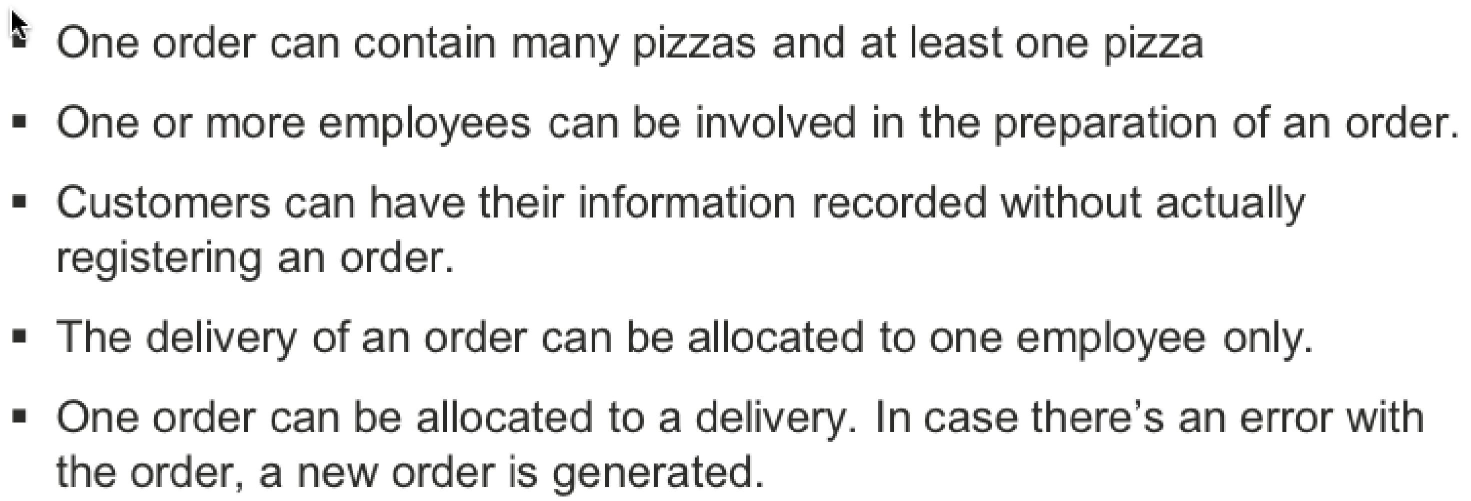
- Personal information
- Address
- Phone Number

- Pizza ordered
- Quantity
- Delivery address
- Order time
- Order status

- Delivery status
- Delivery time
- Allocated vehicle
- Allocated Employee

Case Study

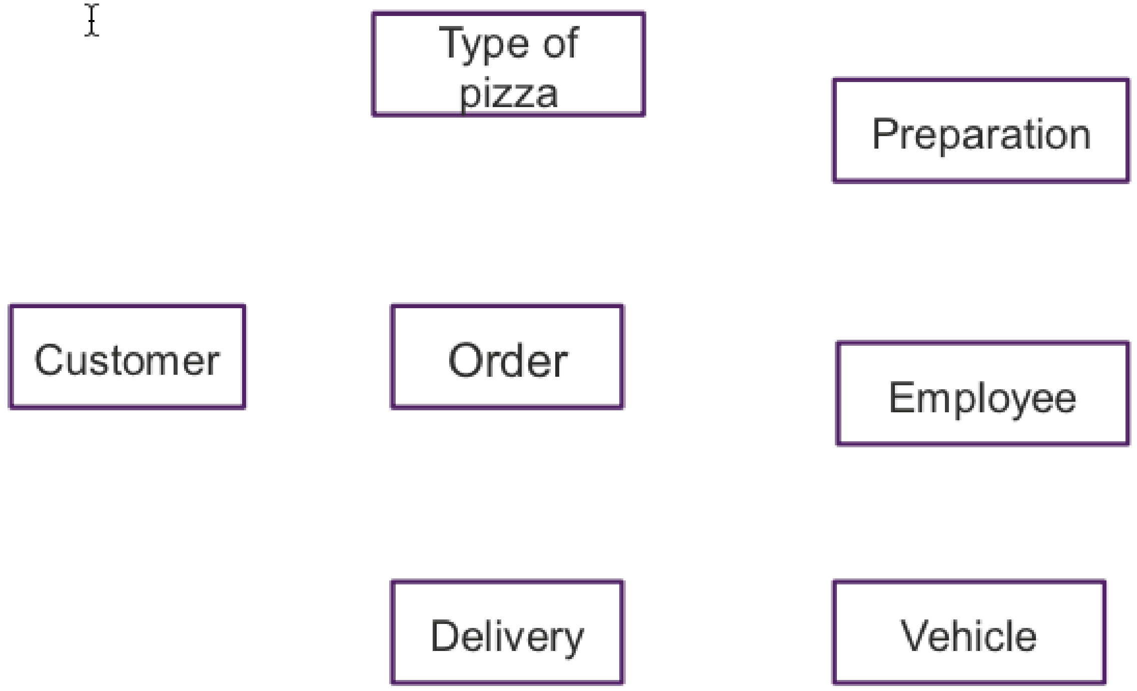
Understand the business



- One order can contain many pizzas and at least one pizza
- One or more employees can be involved in the preparation of an order.
- Customers can have their information recorded without actually registering an order.
- The delivery of an order can be allocated to one employee only.
- One order can be allocated to a delivery. In case there's an error with the order, a new order is generated.

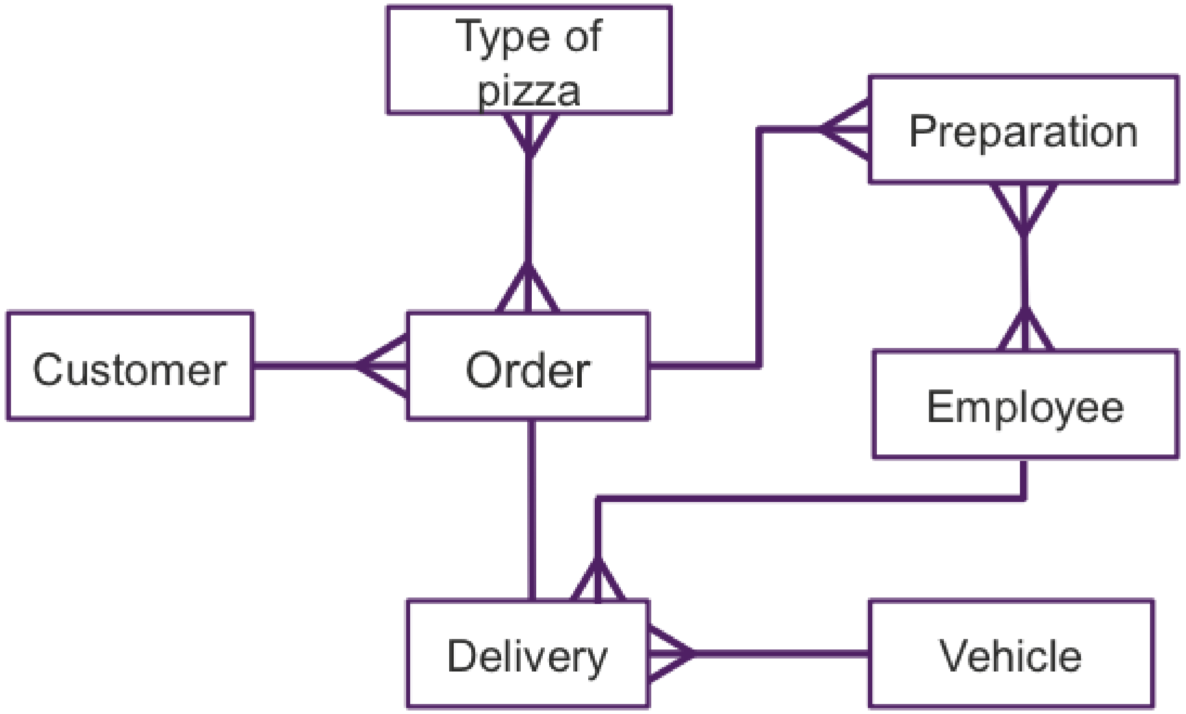
Case Study

Identify entities and put them on paper



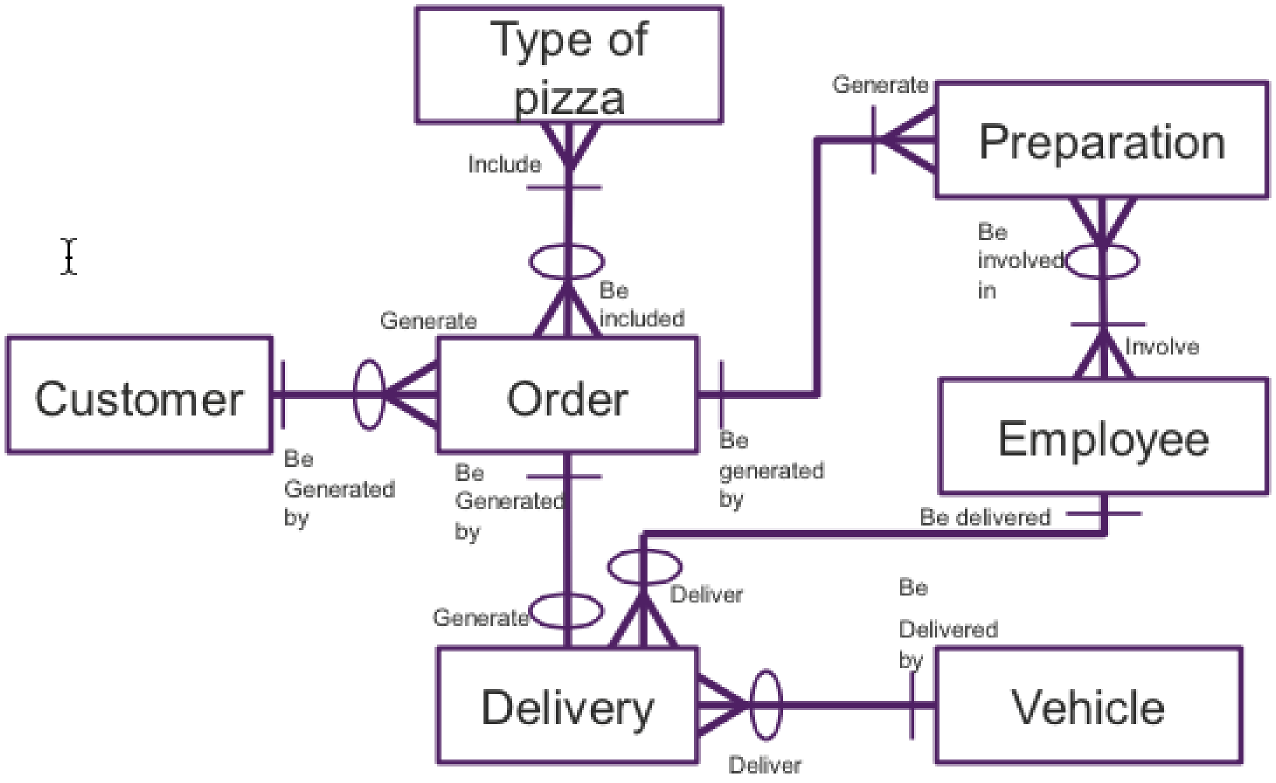
Case Study

Identify relationships



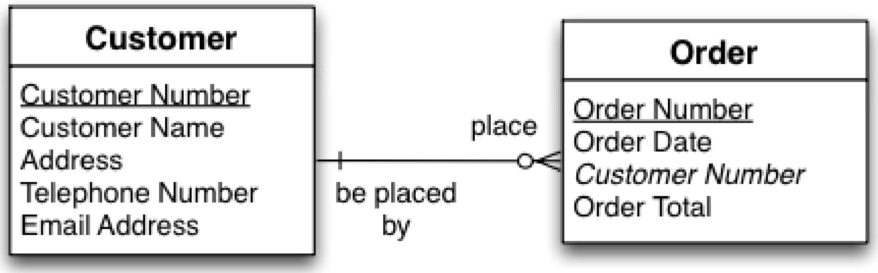
Case Study

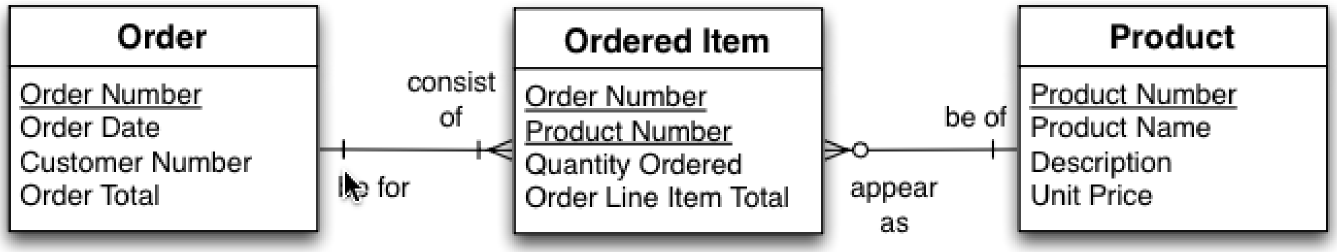
Identify Optionality



Case Study

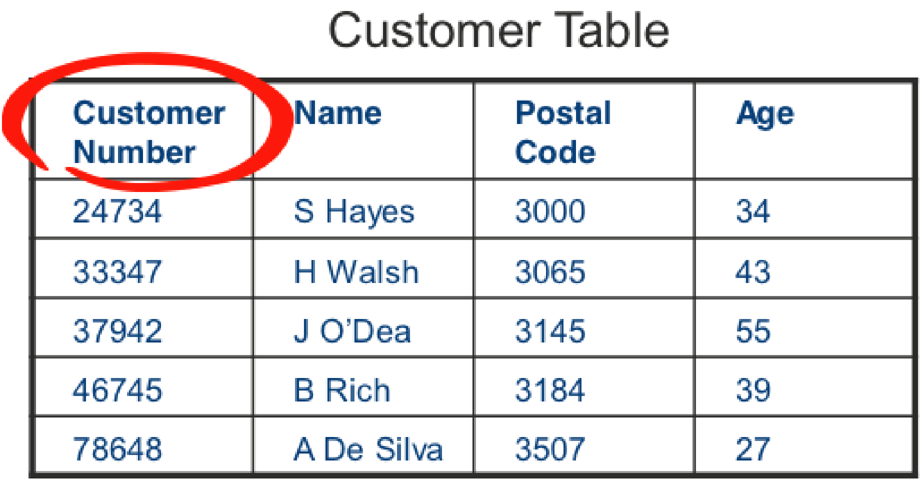
Attributes and keys





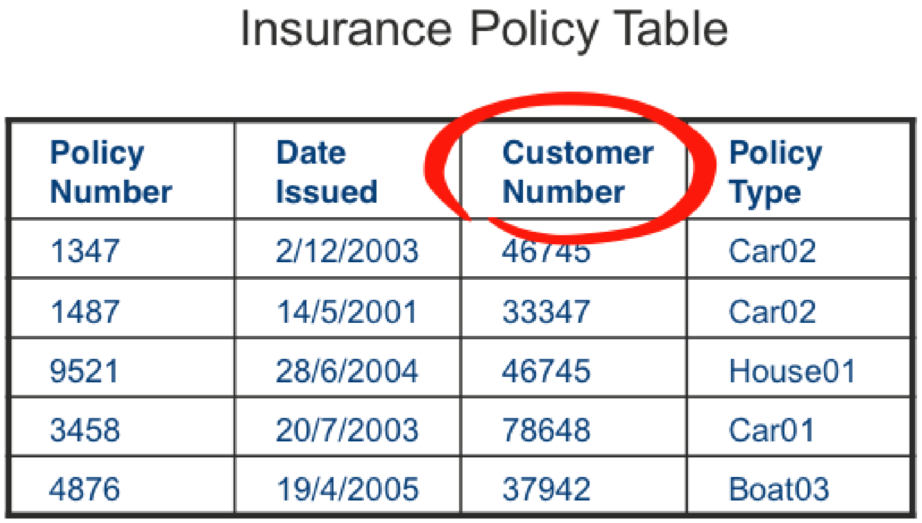
Case Study

Physical implementation



Customer Table

Customer Number	Name	Postal Code	Age
24734	S Hayes	3000	34
33347	H Walsh	3065	43
37942	J O'Dea	3145	55
46745	B Rich	3184	39
78648	A De Silva	3507	27

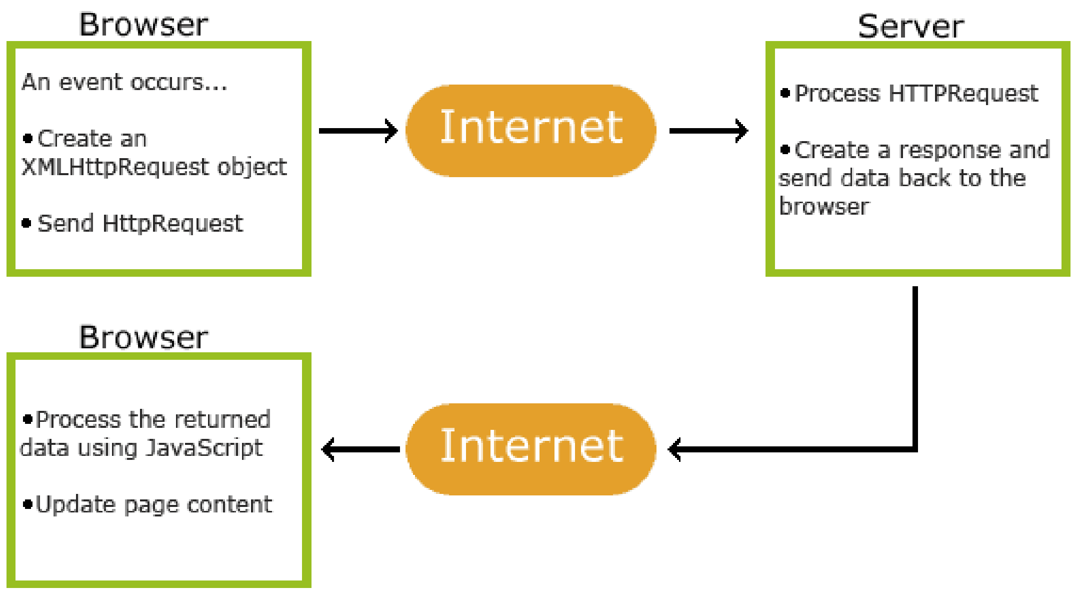


Insurance Policy Table

Policy Number	Date Issued	Customer Number	Policy Type
1347	2/12/2003	46745	Car02
1487	14/5/2001	33347	Car02
9521	28/6/2004	46745	House01
3458	20/7/2003	78648	Car01
4876	19/4/2005	37942	Boat03

AJAX

- AJAX is about updating parts of a web page, without reloading the whole page.
- AJAX = Asynchronous JavaScript and XML.
- AJAX is a technique for creating fast and dynamic web pages.
- AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
- Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Internet
Browser
•Process the returned data using JavaScript
•Update page content

XMLHttpRequest

- XMLHttpRequest is a JavaScript object that provides an easy way to retrieve data from a URL without having to do a full page refresh.
- A Web page can update just a part of the page without disrupting what the user is doing. XMLHttpRequest is used heavily in AJAX programming.
- Despite its name, XMLHttpRequest can be used to retrieve any type of data, not just XML, and it supports protocols other than HTTP (including file and ftp).

```
var myRequest = new XMLHttpRequest();
```

Methods

❖ **open()**

Initializes a request. This method is to be used from JavaScript code.

```
void open(DOMString method, DOMString url, optional boolean async, optional DOMString?
user, optional DOMString? password);

xmlhttp.open("GET", "nameHint.php?q=" + str, true);
```

❖ **send()**

Sends the request. If the request is asynchronous (which is the default), this method returns as soon as the request is sent. If the request is synchronous, this method doesn't return until the response has arrived.

```
void send();
void send(ArrayBufferView data);
void send(Blob data);
void send(Document data);
void send(DOMString? data);
void send(FormData data);
```

Note: *Any event listeners you wish to set must be set before calling* send().

Properties

- XMLHttpRequest.onreadystatechange

  Returns a EventHandler that is called whenever the readyState attribute changes. The callback is called from the user interface thread.

- XMLHttpRequest.readyState

Value	State	Description
0	UNSENT	open() has not been called yet.
1	OPENED	send() has been called.
2	HEADERS_RECEIVED	send() has been called, and headers and status are available.
3	LOADING	Downloading; responseText holds partial data.
4	DONE	The operation is complete.

Properties

- XMLHttpRequest.status

  Returns an unsigned short with the status of the response of the request. This is the HTTP result code (for example, status is 200 for a successful request).

- XMLHttpRequest.statusText

  Returns a DOMString containing the response string returned by the HTTP server. Unlike XMLHTTPRequest.status, this includes the entire text of the response message ("200 OK", for example).

Properties

- XMLHttpRequest.response

  Returns an ArrayBuffer, Blob, Document, JavaScript object, or a DOMString, depending of the value of XMLHttpRequest.responseType. that contains the response entity body. This is null if the request is not complete or was not successful.

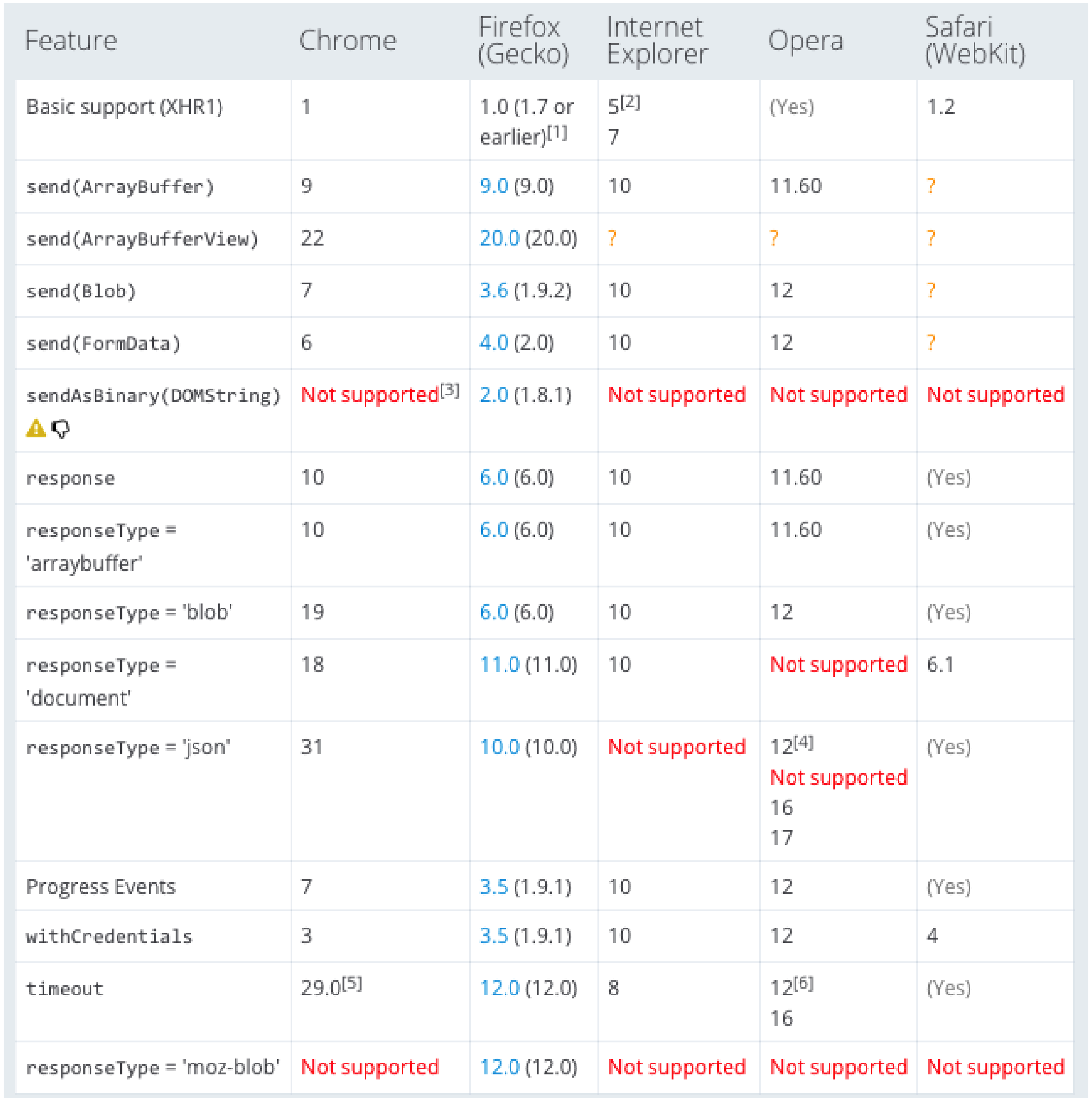
- XMLHttpRequest.responseText

  Returns a DOMString that contains the response to the request as text, or null if the request was unsuccessful or has not yet been sent.

- XMLHttpRequest.responseType

  Is an enumerated value that defines the response type. It can have the following values:””(DOMString (this is the default value)), "arraybuffer”, "document”, "json”, "text"

Browser compatibility



Feature	Chrome	Firefox (Gecko)	Internet Explorer	Opera	Safari (WebKit)
Basic support (XHR1)	1	1.0 (1.7 or earlier)[1]	5[2] 7	(Yes)	1.2
`send(ArrayBuffer)`	9	9.0 (9.0)	10	11.60	?
`send(ArrayBufferView)`	22	20.0 (20.0)	?	?	?
`send(Blob)`	7	3.6 (1.9.2)	10	12	?
`send(FormData)`	6	4.0 (2.0)	10	12	?
`sendAsBinary(DOMString)` ⚠ 👎	Not supported[3]	2.0 (1.8.1)	Not supported	Not supported	Not supported
`response`	10	6.0 (6.0)	10	11.60	(Yes)
`responseType` = 'arraybuffer'	10	6.0 (6.0)	10	11.60	(Yes)
`responseType` = 'blob'	19	6.0 (6.0)	10	12	(Yes)
`responseType` = 'document'	18	11.0 (11.0)	10	Not supported	6.1
`responseType` = 'json'	31	10.0 (10.0)	Not supported	12[4] Not supported 16 17	(Yes)
Progress Events	7	3.5 (1.9.1)	10	12	(Yes)
`withCredentials`	3	3.5 (1.9.1)	10	12	4
`timeout`	29.0[5]	12.0 (12.0)	8	12[6] 16	(Yes)
`responseType` = 'moz-blob'	Not supported	12.0 (12.0)	Not supported	Not supported	Not supported