**Database**

- structured set of data held in a computer, especially one that is accessible in various ways

- is basically an organized collection of information/ data in such a way that a [computer](http://www.webopedia.com/TERM/C/computer.html) [program](http://www.webopedia.com/TERM/P/program.html) can quickly [select](http://www.webopedia.com/TERM/S/select.html) desired pieces of [data](http://www.webopedia.com/TERM/D/data.html). You can think of a database as an electronic filing [system](http://www.webopedia.com/TERM/S/system.html).

- is an organized collection of [data](https://en.wikipedia.org/wiki/Data_(computing)). It is the collection of [schemas](https://en.wikipedia.org/wiki/Database_schema), [tables](https://en.wikipedia.org/wiki/Table_(database)), [queries](https://en.wikipedia.org/wiki/Query_language), reports, [views](https://en.wikipedia.org/wiki/View_(SQL)), and other objects. The data are typically organized to model aspects of reality in a way that supports [processes](https://en.wikipedia.org/wiki/Process_(computing)) requiring information,

**DBMS**

- Database Management System

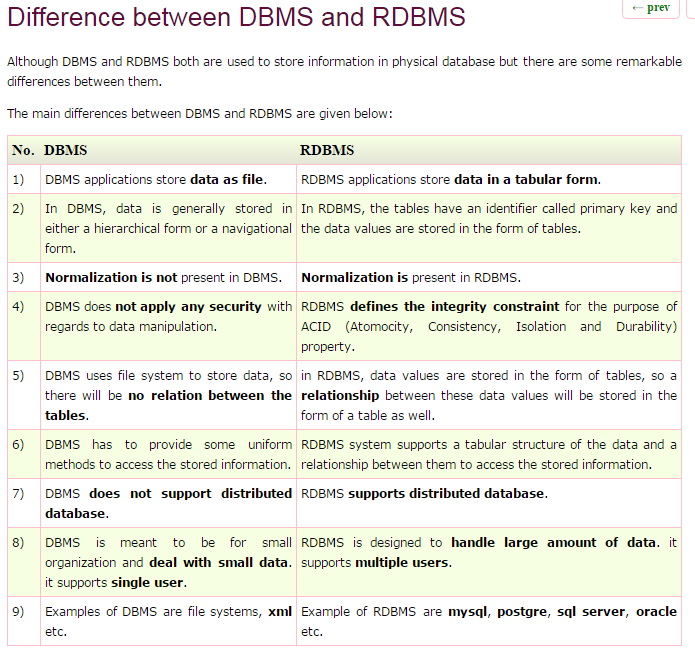
- enables you to store and [access](http://www.webopedia.com/TERM/A/access.html) information from a database in such a way that it becomes easier to retrieve, manipulate, and produce information.

**RDBMS**

- Relational Database Management System

- The RDBMS database uses tables to store data. A table is a collection of related data entries and contains rows and columns to store data.

- All relational database management systems like MySQL, MS Access, Oracle, Postgres and SQL Server use SQL as standard database language.



**SQL**

- Structured Query Language

- is a computer language used to perform operations like storing, manipulating and retrieving of data stored in relational database.

- is the standard language for Relation Database System.

DDL - Data Definition Language:

|  |  |
| --- | --- |
| **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. |
| TRUNCATE | Remove all records from a table, including all spaces allocated for the records are removed |

DML - Data Manipulation Language:

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

DCL - Data Control Language:

|  |  |
| --- | --- |
| **Command** | **Description** |
| GRANT | Gives a privilege to user |
| REVOKE | Takes back privileges granted from user |

**SQL Joins**

- to combine two or more tables.

- the SQL JOIN clause takes records from two or more tables in a database and combines it together.

**5 types of JOIN (ANSI standard SQL):**

Scenario: Lets say you have a Students table, and a Lockers table.

Each student can be assigned to a locker, so there is a Locker Number column in the Student table. More than one student could potentially be in a single locker, but especially at the beginning of the school year, you may have some incoming students without lockers and some lockers that have no students assigned.

For the sake of this example, let’s say you have 100 students, 70 of which have lockers. You have a total of 50 lockers, 40 of which have at least 1 student and 10 lockers have no student.

**1. INNER JOIN (JOIN)**

- Returns matched records from both tables. Selects all rows from both tables as long as there is a match between the columns.

Scenario: "show me all students with lockers".  
Any students without lockers, or any lockers without students are missing.  
Returns 70 rows

**2. LEFT JOIN (LEFT OUTER JOIN)**

- this join returns all the rows from left table combine with the matching rows of the right table. If you get no matching in the right table it returns NULL values.

Scenario: "show me all students, with their corresponding locker if they have one".   
This might be a general student list, or could be used to identify students with no locker.   
Returns 100 rows

**3. RIGHT JOIN (RIGHT OUTER JOIN)**

- this join returns all the rows from right table are combined with the matching rows of left table .If you get no column matching in the left table .it returns null value.

Scenario: "show me all lockers, and the students assigned to them if there are any".   
This could be used to identify lockers that have no students assigned, or lockers that have too many students.   
Returns 80 rows (list of 70 students in the 40 lockers, plus the 10 lockers with no student)

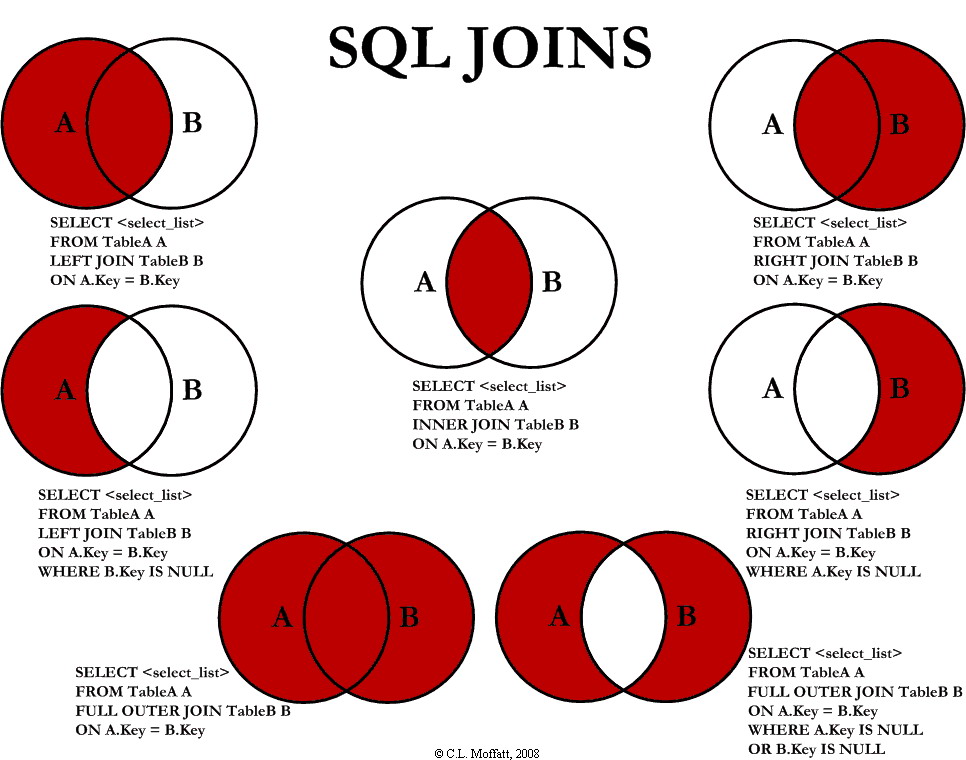
**4. FULL JOIN (FULL OUTER JOIN)**

- is used to combine the result of both left and right outer join and returns all rows (don’t care if it’s matched or unmatched) from the both participating tables.  
Scenario: "show me all students and all lockers, and match them up where you can"   
Returns 110 rows (all 100 students, including those without lockers. Plus the 10 lockers with no student)

**5. CROSS JOIN**

- When each row of first table is combined with each row from the second table, known as Cartesian join. It returns the Cartesian product of the sets of rows from the joined table.

Scenario: It doesn't use the linked locker number field in the students table, so you basically end up with a big giant list of every possible student-to-locker pairing, whether or not it actually exists.  
Returns 5000 rows (100 students x 50 lockers). Could be useful (with filtering) as a starting point to match up the new students with the empty lockers.



**SQL Constraints**

SQL constraints are used to specify rules for the data in a table. If there is any violation between the constraint and the data action, the action is aborted by the constraint.

SQL constraint (rule/ restriction) that must remain true for a database to preserve data integrity. These are specified at database creation time and enforce by the DBMS

**Primary Key**

- column that uniquely identifies each row in a table

**Foreign Key**

- points to a primary key in another table. Prevents invalid data from being inserted into the foreign key column. The values that you enter into the foreign key column, has to be one of the values contained in the table it points to. Foreign key column allows only values if they exist to the referenced primary key column of other table

**Unique**

- a constraint that uniquely identifies each record in a database table.

- the UNIQUE and PRIMARY KEY constraints both provide uniqueness for a column or set of columns. The difference is that you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table

**Not Null**

- enforces a column to NOT accept NULL values. Meaning, you cannot insert a new record, or update a record without adding a value to this field

**DEFAULT**

- is used to insert a default value into a column. The default value will be added to all new records, if no other value is specified

**CHECK**

- used to limit the value range that can be placed in a column. Meaning, there are only certain values that you can set in column

**OTHER SQL Keys**

**Super Keys**

-  is a set or one of more columns (attributes) to uniquely identify rows in a table.

**Candidate Keys**

- Candidate keys are selected from the set of super keys, the only thing we take care while selecting candidate key is: It should not have any redundant attribute.

**Alternative Keys**

- Out of all [candidate keys](http://beginnersbook.com/2015/04/candidate-key-in-dbms/), only one gets selected as [primary key](http://beginnersbook.com/2015/04/primary-key-in-dbms/), remaining keys are known as alternative.

**Composite Keys**

- When multiple columns are used as a primary key, it is known as composite key. A composite key is a combination of two or more columns in a table that can be used to uniquely identify each row in the table

- A key that consists of more than one attribute to uniquely identify rows (also known as records & tuples) in a table is called composite key.

Let’s take an example to understand this: Employee table

|  |  |  |
| --- | --- | --- |
| **Emp\_Number** | **Emp\_SSN** | **Emp\_Name** |
| 226 | 123456789 | Steve |
| 227 | 999999321 | Ajeet |
| 228 | 888997212 | Chaitanya |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Super Keys** | **Alternative Keys / Secondary Keys** | **Primary Key** |
| **Candidate keys / Minimal Super Keys** | {Emp\_SSN}  {Emp\_Number} | {Emp\_SSN} | {Emp\_Number} |
| **Composite Keys / Compound keys** | {Emp\_SSN, Emp\_Number}  {Emp\_SSN, Emp\_Name}  {Emp\_SSN, Emp\_Number, Emp\_Name}  {Emp\_Number, Emp\_Name} | [X] | [X] |

**OTHER TERMS**

**Data Integrity -** refers to maintaining and assuring the accuracy and consistency of data. Data security is the protection data from the unauthorized users.

**QUERY –** primary mechanism for retrieving information from a database. A request information from a database.

**SCHEMA –** is a way to logically group objects such as tables, views, etc. Organization or structure for a database.

**columns** = attributes = fields

**rows** = tuples = records

**DISTINCT –** gives you all the rows, eliminating duplicates from the result set. Used along with SELECT keyword retrieves only unique data entries depending on the column list you have specified after it.

**AUTO\_INCREMENT** (MySQL and access) **/ SERIAL** (postgresql) **/ IDENTITY** (ms sql)

If a column is marked as AUTO\_INCREMENT /SERIAL/ IDENTITY, then the values for this column are automatically generated when you insert a new row into the table.

For ORACLE, you need to create a sequence which will be then used as a value (.nextval) when inserting record.

**Group by –** is used to group a selected set of rows into a set of summary rows by values of one or more columns or expressions. It is always used in conjunction with one or more aggregate functions

**Where Clause –** is used before the aggregation takes place

**Having Clause –** is used to check conditions after the aggregation takes place

**Difference between Where and Having Clause**

1. WHERE clause can be used with – Select, Insert, and Update statements, whereas HAVING clause can only be used with the Select Statement

2. WHERE filters rows before aggregation(GROUPING), whereas HAVING filter groups, after the aggregations are performed

3. Aggregate functions cannot be used in the WHERE clause, unless it is on a sub query contained in a HAVING clause, whereas aggregate functions can be used in Having clause

**UNION and UNION ALL**

-operators in SQL Server, are used to combine the result-set of two or more SELECT queries

-for UNION and UNION ALL to work, the Number, Data Types, and the order of the columns in the select statements should be the same

-ORDER BY clause should be used only on the last SELECT statement in the UNION query

**Difference between TRUNCATE and DELETE**

TRUNCATE DELETE TRUNCATE is a DDL command DELETE is a DML command TRUNCATE is executed using a table lock and whole table is locked for remove all records. DELETE is executed using a row lock, each row in the table is locked for deletion. We cannot use Where clause with TRUNCATE. We can use where clause with DELETE to filter & delete specific records. TRUNCATE removes all rows from a table. The DELETE command is used to remove rows from a table based on WHERE condition. Minimal logging in transaction log, so it is performance wise faster. It maintain the log, so it slower than TRUNCATE. TRUNCATE TABLE removes the data by deallocating the data pages used to store the table data and records only the page deallocations in the transaction log. The DELETE statement removes rows one at a time

**DIFFERENCE BETWEEN UNION AND UNION ALL**

1. UNION removes duplicate rows, whereas UNION ALL does not.

2. UNION has to perform distinct sort to remove duplicates, which makes it less faster than UNION ALL

**DIFFERENCE BETWEEN UNION AND JOIN**

UNION combines the result-set of two or more select queries into a single result-set which includes all the rows from all the queries in the union, whereas JOINS retrieve data from two or more tables based on logical relationships between the tables

In short, UNION combines rows from two or more tables, where JOINS combine columns from 2 or more tables

**SQL INJECTION**

SQL injection is a technique where malicious users can inject SQL commands into an SQL statement, via the input data from the client to the application.

Injected SQL commands can alter SQL statement and compromise the security of a web application.

SQL injection weaknesses occur when an application uses untrusted data, such as data entered into web form fields, as part of a database query. When an application fails to properly sanitize this untrusted data before adding it to a SQL query, an attacker can include their own SQL commands which the database will execute. SQL injection is used by someone to, Obtain information you have in your database and Delete or change information in your database.

Protection

* **Constrain and sanitize input data.** Check for known good data by validating for type, length, format, and range.
* **Use type-safe SQL parameters for data access.** You can use these parameters with stored procedures or dynamically constructed SQL command strings. Parameter collections such as SqlParameterCollection provide type checking and length validation. If you use a parameters collection, input is treated as a literal value, and SQL Server does not treat it as executable code. An additional benefit of using a parameters collection is that you can enforce type and length checks. Values outside of the range trigger an exception. This is a good example of defense in depth.
* **Use an account that has restricted permissions in the database.** Ideally, you should only grant execute permissions to selected stored procedures in the database and provide no direct table access.
* **Avoid disclosing database error information.** In the event of database errors, make sure you do not disclose detailed error messages to the user.

**ACID**

- Atomicity, Consistency, Isolation, Durability

- A transaction in a database system must maintain the ACID properties − in order to ensure accuracy, completeness, and data integrity.  A transaction is a sequence of operations performed as a single logical unit of work.

- These are important properties of a database system’s architecture. Specifically these properties refer to how database transactions are designed.

**Atomicity**

When a database processes a transaction, it can either be all committed or did nothing at all. The database considers all transaction operations as one whole unit or atom, so if one part of the transaction fails, the entire transaction fails. This ensures that the database is in a valid state at all times.

Scenario: A transaction to transfer funds from one account to another involves making a withdrawal operation from the first account and a deposit operation on the second. If the deposit operation failed, you don’t want the withdrawal operation to happen either.

**Consistency**

Ensures that only valid data will be written to the database. If, for some reason, a transaction is executed that violates the database’s rules and constraints, the entire transaction will be rolled back and the database will be restored to a state consistent with those rules.

Scenario: A database tracking a checking account may only allow unique check numbers to exist for each transaction. So whenever a teller/ user enters an existing cheque number the database will validate it and if the cheque number exists then the database should not accept the transaction.

**Isolation**

Ensures that each transaction appears to be the only transaction manipulating the data, even though other transaction may be running at the same time, but it does not ensure the order of transactions.

The database system can either provide a prior view during a separate concurrent transaction, or it can “lock” the resource until the transaction completes.

Scenario: A teller looking up a balance must be isolated from a concurrent transaction involving a withdrawal from the same account. Only when the withdrawal transaction commits successfully and the teller looks at the balance again will the new balance be reported.

**Durability**

Ensures that once a transaction is complete, the information must be saved permanently to the database and will not be lost in case of failures of any kind. When designing a database system, only two kinds of malfunctions are considered: media failure and system failure. For media failure (e.g. a faulty hard drive) databases are recovered by using backups and transaction logs. And System failures (e.g. system crashes, power outages etc…) have to be handled too.

Scenario: When withdrawing cash in an ATM and it’s about to take out the money when suddenly the system shuts down. Transaction should not be fully committed/completed, since ATM didn’t took out any money, and once you check your account balance again, the record should return to its consistent state before the failed transaction.

**NORMALIZATION**

- is a process of organizing the data in database to avoid data redundancy, insertion, update & deletion anomaly

|  |  |
| --- | --- |
| **ADVANTAGES OF NORMALIZATION**  ● More efficient data structure.  ● Avoid redundant fields or columns.  ● More flexible data structure i.e. we should be able to add new rows and data values easily  ● Better understanding of data.  ● Ensures that distinct tables exist when necessary.  ● Easier to maintain data structure i.e. it is easy to perform operations and complex queries can be easily handled. ● Minimizes data duplication. ● Close modeling of real world entities, processes and their relationships. | **DISADVANTAGES OF NORMALIZATION**  ● You cannot start building the database before you know what the user needs. ● On Normalizing the relations to higher normal forms i.e. 4NF, 5NF the performance degrades. ● It is very time consuming and difficult process in normalizing relations of higher degree. ● Careless decomposition may leads to bad design of database which may leads to serious problems. |

**First normal form (1NF)**

- an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.

Scenario: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

|  |  |  |  |
| --- | --- | --- | --- |
| **emp\_id** | **emp\_name** | **emp\_address** | **emp\_mobile** |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212  9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123  8123450987 |

Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.

This table is not in 1NF as the rule says “each attribute of a table must have atomic (single) values”, the emp\_mobile values for employees Jon & Lester violates that rule.

To make the table complies with 1NF we should have the data like this:

|  |  |  |  |
| --- | --- | --- | --- |
| **emp\_id** | **emp\_name** | **emp\_address** | **emp\_mobile** |
| 101 | Herschel | New Delhi | 8912312390 |
| 102 | Jon | Kanpur | 8812121212 |
| 102 | Jon | Kanpur | 9900012222 |
| 103 | Ron | Chennai | 7778881212 |
| 104 | Lester | Bangalore | 9990000123 |
| 104 | Lester | Bangalore | 8123450987 |

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

**Second normal form (2NF)**

A table is said to be in 2NF if both the following conditions hold:

1. Table is in 1NF (First normal form)

2. No non-prime attribute is dependent on the proper subset of any candidate key of table. There must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails Second normal form.

**Prime attribute** − An attribute, which is a part of the prime-key, is known as a prime attribute.

**Non-prime attribute** − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

Scenario: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

|  |  |  |
| --- | --- | --- |
| **teacher\_id** | **subject** | **teacher\_age** |
| 111 | Maths | 38 |
| 111 | Physics | 38 |
| 222 | Biology | 38 |
| 333 | Physics | 40 |
| 333 | Chemistry | 40 |

Candidate Keys: {teacher\_id, subject}  
Non prime attribute: teacher\_age

The table is in 1 NF because each attribute has atomic values.

However, it is not in 2NF because there are two rows for teacher ids 111 to include multiple subjects they handled. While this is searchable, it is an inefficient use of space. Also in the above Table, while the candidate keys {teacher\_id, subject}, teacher\_age only depends on teacher\_id column, which is incorrect as per Second Normal Form.

To achieve second normal form, it would be helpful to split out the subjects into an independent table, and match them up using the teacher\_id as foreign keys.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| teacher\_details table:   |  |  | | --- | --- | | **teacher\_id** | **teacher\_age** | | 111 | 38 | | 222 | 38 | | 333 | 40 | | teacher\_subject table:   |  |  | | --- | --- | | **teacher\_id** | **subject** | | 111 | Maths | | 111 | Physics | | 222 | Biology | | 333 | Physics | | 333 | Chemistry | |

Now the tables comply with Second normal form (2NF).

**Third normal form (3NF)**

A table design is said to be in 3NF if both the following conditions hold:

1. Table must be in 2NF

2. [Transitive functional dependency](http://beginnersbook.com/2015/04/transitive-dependency-in-dbms/) of non-prime attribute on any super key should be removed.

An attribute that is not part of any [candidate key](http://beginnersbook.com/2015/04/candidate-key-in-dbms/) is known as non-prime attribute.

Scenario: Suppose a company wants to store the complete address of each employee, they create a table named employee\_details that looks like this:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **emp\_id** | **emp\_name** | **emp\_zip** | **emp\_state** | **emp\_city** | **emp\_district** |
| 1001 | John | 282005 | UP | Agra | Dayal Bagh |
| 1002 | Ajeet | 222008 | TN | Chennai | M-City |
| 1006 | Lora | 282007 | TN | Chennai | Urrapakkam |
| 1101 | Lilly | 292008 | UK | Pauri | Bhagwan |
| 1201 | Steve | 222999 | MP | Gwalior | Ratan |

In this table emp\_id is Primary key, but emp\_state, emp\_city and emp\_district depends upon emp\_zip. The dependency between emp\_zip and other fields is called transitive dependency. Hence to apply 3NF, we need to move the emp\_state, emp\_city and emp\_district to new table, with emp\_zip as primary key.

To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| employee table:   |  |  |  | | --- | --- | --- | | **emp\_id** | **emp\_name** | **emp\_zip** | | 1001 | John | 282005 | | 1002 | Ajeet | 222008 | | 1006 | Lora | 282007 | | 1101 | Lilly | 292008 | | 1201 | Steve | 222999 | | employee\_zip table:   |  |  |  |  | | --- | --- | --- | --- | | **emp\_zip** | **emp\_state** | **emp\_city** | **emp\_district** | | 282005 | UP | Agra | Dayal Bagh | | 222008 | TN | Chennai | M-City | | 282007 | TN | Chennai | Urrapakkam | | 292008 | UK | Pauri | Bhagwan | | 222999 | MP | Gwalior | Ratan | |

The advantage of removing transitive dependency is,

1. Amount of data duplication is reduced.

2. Data integrity achieved.

**Boyce & Codd normal form (BCNF)**

- is a higher/ advanced version of the Third Normal form that’s why it is also referred as 3.5NF. BCNF is stricter than 3NF. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF.

Scenario: Suppose there is a company wherein employees work in more than one department. They store the data like this:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **emp\_id** | **emp\_nationality** | **emp\_dept** | **dept\_type** | **dept\_no\_of\_emp** |
| 1001 | Austrian | Production and planning | D001 | 200 |
| 1001 | Austrian | stores | D001 | 250 |
| 1002 | American | design and technical support | D134 | 100 |
| 1002 | American | Purchasing department | D134 | 600 |

Functional dependencies in the table above:  
emp\_id -> emp\_nationality  
emp\_dept -> {dept\_type, dept\_no\_of\_emp}

Candidate key: {emp\_id, emp\_dept}

The table is not in BCNF as neither emp\_id nor emp\_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:  
emp\_nationality table:

|  |  |
| --- | --- |
| **emp\_id** | **emp\_nationality** |
| 1001 | Austrian |
| 1002 | American |

emp\_dept table:

|  |  |  |
| --- | --- | --- |
| **emp\_dept** | **dept\_type** | **dept\_no\_of\_emp** |
| Production and planning | D001 | 200 |
| stores | D001 | 250 |
| design and technical support | D134 | 100 |
| Purchasing department | D134 | 600 |

emp\_dept\_mapping table:

|  |  |
| --- | --- |
| **emp\_id** | **emp\_dept** |
| 1001 | Production and planning |
| 1001 | stores |
| 1002 | design and technical support |
| 1002 | Purchasing department |

Functional dependencies:  
emp\_id -> emp\_nationality  
emp\_dept -> {dept\_type, dept\_no\_of\_emp}

Candidate keys:  
For first table: emp\_id  
For second table: emp\_dept  
For third table: {emp\_id, emp\_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.