Релационни база данни

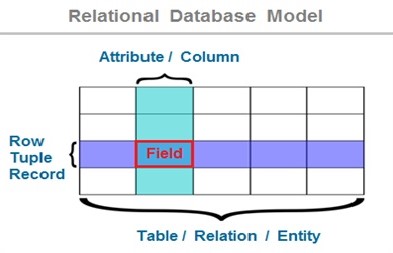
1. 7 нормални форми

**Релация**- представлява препокриващо съпоставяне между елементи от две или повече множества.

Видове релация:

1. рефлексивна
2. антирефлексивна
3. симетрична
4. антисиметрична
5. силно антисиметрична
6. транзитивна

**Релационна база данни**- тип база данни, която съхранява множество данни във вид на релации, съставени от записи и атрибути (полета) и възприемани от потребителите като таблици.



* *релация*, *релационна схема* (relation) ↔ *таблица* (table),
* *запис*, *кортеж* (tuple) ↔ *ред* (row),
* *атрибут*, *поле* (attribute) ↔ *стълб*, *колона* (column).

**Normalization** is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updating anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

**Нормална форма**- (използва се за елиминиране или намаляване на излишъка в таблиците на базата данни )

**Redundancy**- съкращения, излишък

**Set of**- набор от…, комплект

1. **First Normal Form**

If a relation contains composite or multi-valued attribute, it violates first normal form or a relation is in first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

**Първа нормална норма-** Релацията е в първа нормална форма, ако всеки атрибут в тази връзка е атрибут със стойност.

**Multi-valued**- многозначен

**Composite**- съставен

**Violates**- нарушава

|  |  |  |
| --- | --- | --- |
| ID Name Courses  ------------------  1 A c1, c2  2 E c3  3 M C2, c3 | Example for First Normal Form | ID Name Course  ------------------  1 A c1  1 A c2  2 E c3  3 M c2   1. M c3 |

1. **Second Normal Form**

To be in second normal form, a relation must be in first normal form and relation must not contain any partial dependency. A relation is in 2NF if it has **No Partial Dependency,**i.e.**,**no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

**[**За да бъде във втора нормална форма, връзката трябва да бъде в първата нормална форма и връзката не трябва да съдържа частична зависимост. Релацията е в 2NF, ако няма частична зависимост, т.е. нито един не-главен атрибут (атрибути, които не са част от нито един кандидат-ключ) не зависи от правилното подмножество, на който и да е кандидат-ключ на таблицата.**]**

**Partial**- частичен

**Dependency**- зависимост

**Subset**- подмножество

**Подобно на следствие, няколко последователни стъпки, които са част от едно действие**.

|  |
| --- |
| STUD\_NO COURSE\_NO COURSE\_FEE  1 C1 1000  2 C2 1500  1 C4 2000  4 C3 1000  4 C1 1000  2 C5 2000  COURSE\_FEE cannot alone decide the value of COURSE\_NO or STUD\_NO; COURSE\_FEE together with STUD\_NO cannot decide the value of COURSE\_NO; COURSE\_FEE together with COURSE\_NO cannot decide the value of STUD\_NO; Hence, COURSE\_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD\_NO, COURSE\_NO} ; But, COURSE\_NO -> COURSE\_FEE , i.e., COURSE\_FEE is dependent on COURSE\_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE\_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.  To convert the above relation to 2NF, we need to split the table into two tables such as: Table 1: STUD\_NO, COURSE\_NO Table 2: COURSE\_NO, COURSE\_FEE |

|  |  |
| --- | --- |
| Table 1 | Table 2 |
| STUD\_NO COURSE\_NO  1 C1  2 C2  1 C4  4 C3  4 C1 | COURSE\_NO COURSE\_FEE  C1 1000  C2 1500  C3 1000  C4 2000  C5 2000 |

### Third Normal Form

### A relation is in third normal form, if there is no transitive dependency for non-prime attributes as well as it is in second normal form. A relation is in 3NF if **at least one of the following condition holds** in every non-trivial function dependency X –> Y

### Transitive- преходен

### Non- trivial- нетривиално

### image5Example 1

|  |
| --- |
| In relation STUDENT given in Table 4,  FD set: {STUD\_NO -> STUD\_NAME, STUD\_NO -> STUD\_STATE, STUD\_STATE -> STUD\_COUNTRY, STUD\_NO -> STUD\_AGE} Candidate Key: {STUD\_NO}  For this relation in table 4, STUD\_NO -> STUD\_STATE and STUD\_STATE -> STUD\_COUNTRY are true. So STUD\_COUNTRY is transitively dependent on STUD\_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_COUNTRY\_STUD\_AGE) as: STUDENT (STUD\_NO, STUD\_NAME, STUD\_PHONE, STUD\_STATE, STUD\_AGE) STATE\_COUNTRY (STATE, COUNTRY) |

**Example 2**







## **BCNF (Boyce-Codd Normal Form)**

Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one **Candidate**Key.

Sometimes is BCNF is also referred as **3.5 Normal Form.**

## **Fourth Normal Form**

If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.

## **Fifth Normal Form**

A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

## **Sixth Normal Form (Proposed)**

6th Normal Form is not standardized, yet however, it is being discussed by database experts for some time. Hopefully, we would have a clear & standardized definition for 6th Normal Form in the near future...

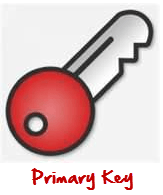
1. Kyes

## What is a KEY in SQL?

A **KEY in SQL** is a value used to identify records in a table uniquely. An SQL KEY is a single column or combination of multiple columns used to uniquely identify rows or tuples in the table. SQL Key is used to identify duplicate information, and it also helps establish a relationship between multiple tables in the database.

Note: Columns in a table that are NOT used to identify a record uniquely are called non-key columns.

(SQL ключът се използва за идентифициране на дублирана информация и също така помага да се установи връзка между множество таблици в базата данни)

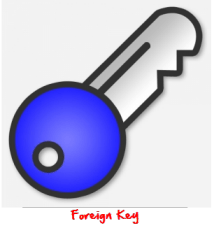


1. Primary KEY

A primary is a single column value used to identify a database record uniquely.

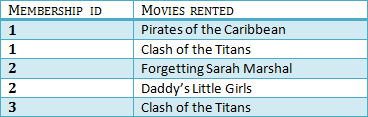
It has following attributes

* A primary key cannot be NULL
* A primary key value must be unique
* The primary key values should rarely be changed
* The primary key must be given a value when a new record is inserted.



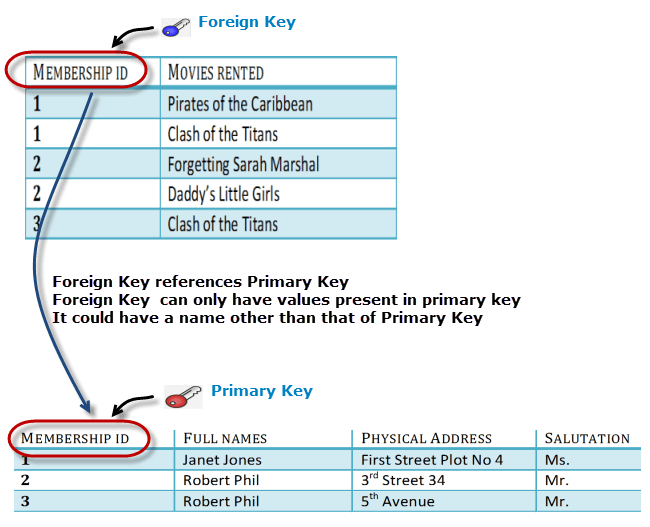
## Foreign Key

In Table 2, Membership\_ID is the Foreign Key

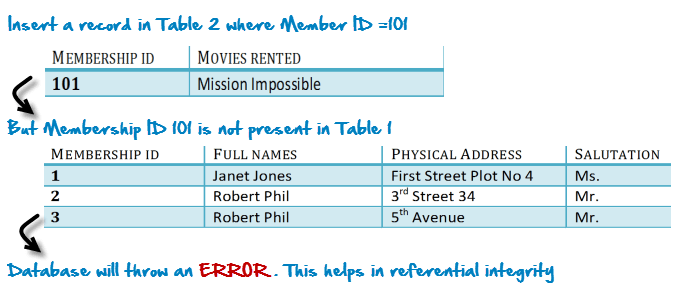


Foreign Key references the primary key of another Table! It helps connect your Tables

* A foreign key can have a different name from its primary key
* It ensures rows in one table have corresponding rows in another
* Unlike the Primary key, they do not have to be unique. Most often they aren't
* Foreign keys can be null even though primary keys can not



## Why do you need a foreign key?



You will only be able to insert values into your foreign key that exist in the unique key in the parent table. This helps in referential integrity.

The above problem can be overcome by declaring membership id from Table2 as foreign key of membership id from Table1

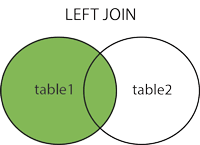
Now, if somebody tries to insert a value in the membership id field that does not exist in the parent table, an error will be shown!

1. Joins

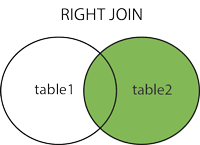
A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

* (INNER) JOIN: Returns records that have matching values in both tables

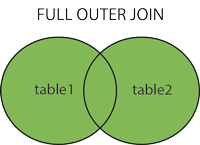
|  |
| --- |
| SELECT column\_name(s) FROM table1 INNER JOIN table2ON table1.column\_name = table2.column\_name; |

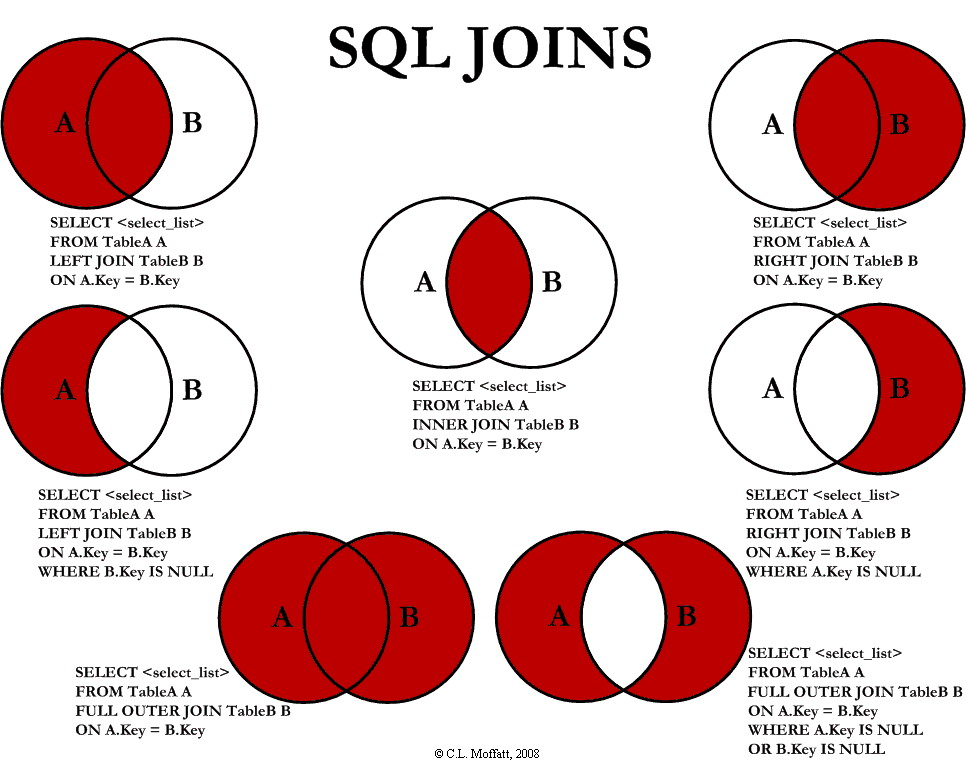
* LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
* **Note:** The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

|  |
| --- |
| SELECT column\_name(s) FROM table1 LEFT JOIN table2ON table1.column\_name = table2.column\_name; |

* RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
* **Note:** The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

|  |
| --- |
| SELECT column\_name(s) FROM table1 RIGHT JOIN table2ON table1.column\_name = table2.column\_name; |



* FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table
  + FULL OUTER JOIN can potentially return very large result-sets!
  + **Note:** The FULL OUTER JOIN keyword returns all matching records from both tables whether the other table matches or not. So, if there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

|  |
| --- |
| SELECT column\_name(s) FROM table1 FULL OUTER JOIN table2ON table1.column\_name = table2.column\_nameWHERE condition; |