Keys

In the relational model, key is used to organize data logically, so that a specific row can be uniquely identified. A key should not be confused with an index. An index is part of the table's physical structure on-disk. It is used to speed data access when queries are performed. Indices are therefore not a part of the relational model.

Firebird/InterBase® automatically generates an index for primary and foreign key columns. On primary key columns, the index actually enforces the unique constraint required by the relational model. So although all keys have indices, not all indices are keys. Links between tables usually occur on primary and foreign keys, so having an index on these columns ensures maximum performance.

Primary key

A primary key is a column (= simple key) or group of columns (= composite key/compound key) used to uniquely define a data set/row in the table. A primary key should always be defined at the time of defining a new table for each table. If you have a database that does not contain primary keys in all tables, and need to add these subsequently, please refer to Adding primary keys to existing tables below.

Relational theory states that a primary key should be designated for every table. It must be unique, and therefore cannot be NULL. It provides automatic protection against storing multiple values. In fact, without a primary key it is impossible to delete just one of two identical data sets. Each table can have only one designated primary key, although it can have other columns that are defined as UNIQUE and NOT NULL.

A primary key column is nothing other than a unique constraint complemented by a system index and the check constraint NOT NULL. Primary keys are always the preferred index of the Firebird/InterBase® Optimizer.

When a data set is created or changed, Firebird/InterBase® immediately checks the validity of the primary key. If the number already exists, a key violation results, and the storage process is immediately cancelled. Unfortunately Firebird/InterBase® allows tables to be created without a primary key, which is a mistake. Data tables should always be keyed.

Existing primary keys and their system names can be viewed on the IBExpert Table Editor / Constraints page.

It is wise to keep the primary key as short as possible to minimize the amount of disk space required, and to improve performance. IBExpert recommends the use of an autoincrement generator ID number used as an internal primary key for all tables. For example, a simple BIGINT data type generator not influenced in any way by any actual data. They do not need to be visible to the user as they are merely a tool to help the database work more efficiently and increase database integrity. One generator can be used as a source for all primary keys in a database, as the numbers do not need to be consecutive but merely unique. Each time a new data set is inserted, the generator automatically generates an ID number, regardless of the table name, for example, new customer_id = 1, new order_id = 2, new orderline_id = 3, new orderline_id = 4, new customer_id = 5, etc. A further

advantage of such a single autoincrement generator primary key is that the database is perfectly prepared for replication; two or more servers can be connected and their data easily swapped, as the primary keys can be simply defined on both servers, e.g. server 1's generator should start at the value 1000000000 and server 2's at 2000000000 thus avoiding any conflict.

Although this method is unfortunately seldom used in the real world, it should be. Each primary key will only ever appear once in the database, which can be quite important in an OO (object-oriented) framework where there are so many objects floating around. They and you both need some unique identifier for the system to tell you what is behind the number, product, order etc.

Since version 1.5 Firebird allows a USING INDEX subclause to be placed at the end of a primary, unique or foreign key definition. Please refer to the *Firebird 2.0 Language Reference Update* chapter, USING INDEX subclause for further information.

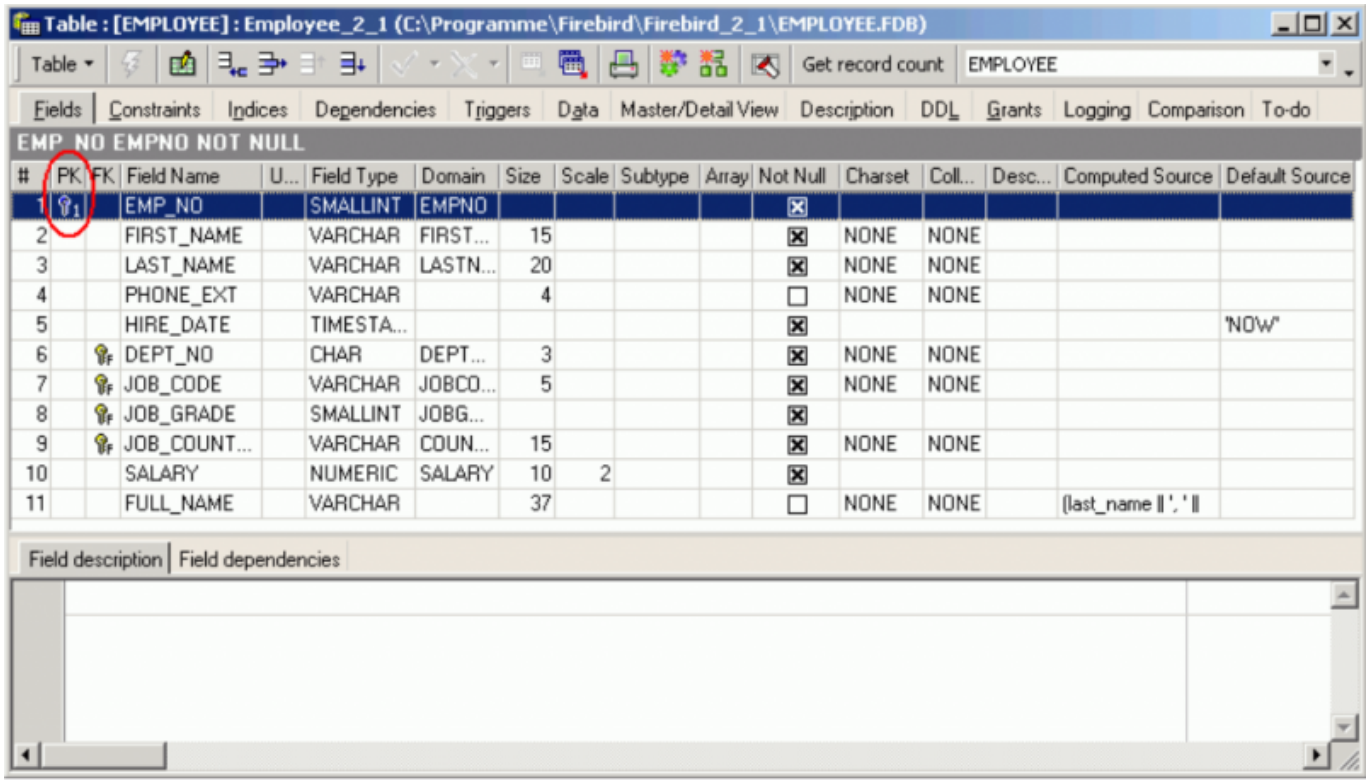
Composite keys are not recommended, as these always slow performance and the sequence of the fields concerned must be identical in all referenced tables.

Adding primary keys to existing tables

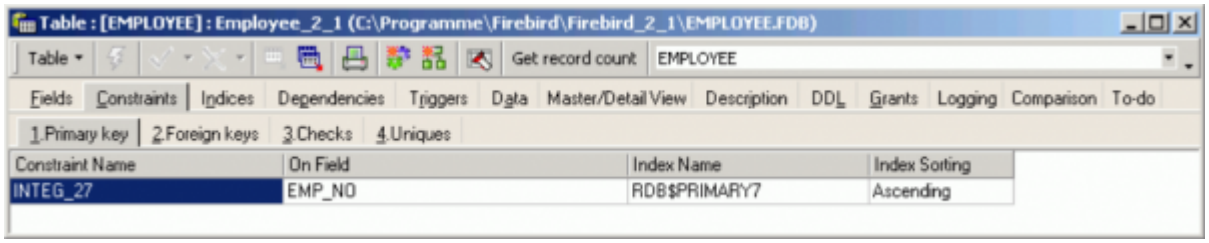
This article was written by Melvin Cox, and provides a method of defining primary keys on existing tables using IBExpert:

Here is a viable workaround for those of us who do not wish to spend an eternity exporting data, dropping and recreating multiple tables, and finally import the data back into those tables. Working with a Firebird 1.5 database (dialect 1) created via ODBC export from a Microsoft Access database, I have successfully defined primary keys on tables by taking the following steps:

1. Bring up the table within the IBExpert interface's Table Editor window (double-click on the respective table in the DB Explorer or use [Ctrl. + O]). The *Fields* page should be active.



2. Double click in the NOT NULL box corresponding to the field that you wish to designate as the primary key. This will call up the *Edit Field* dialog.
3. Check the NOT NULL option and select an existing or create a new domain.



4. Press *OK* and then, after checking the script produced by IBExpert, the *Commit* button. The field is now set to NOT NULL.
5. Bring up the SQL Editor: Tools / SQL Editor (or press [F12]).
6. Enter the following command:

ALTER TABLE table_name ADD PRIMARY KEY (field_name);
For example, to define a primary key on the EVENTS table enter:
ALTER TABLE events ADD PRIMARY KEY (event_id);

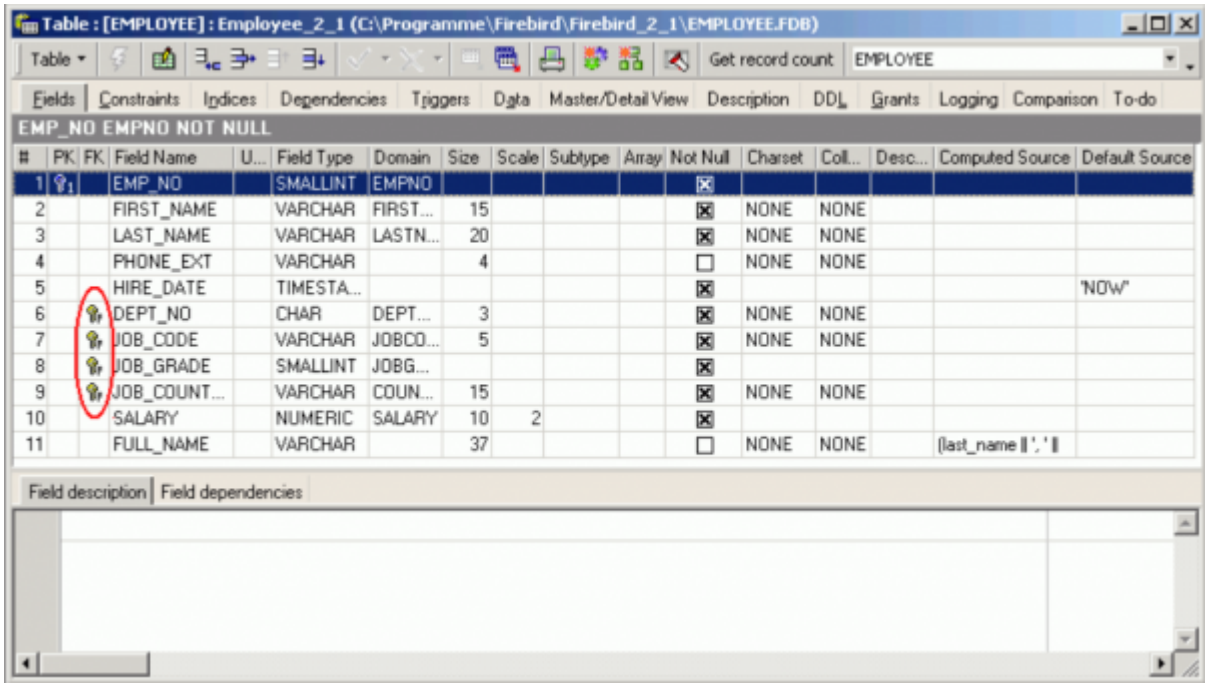
7. Press the *Execute Button* or [F9].
8. Close the SQL Editor. This will call up the *Active Transaction Found* dialog. Select *Commit*.
9. Close the Table Editor window.
10. Reopen the Table Editor window [Ctrl. + O]. The newly defined primary key will now be visible.

back to top of page

Foreign key

A foreign key is composed of one or more columns that reference a primary key. *Reference* means here that when a value is entered in a foreign key, Firebird/InterBase® checks that the value also exists in the referenced primary key. This is used to maintain domain integrity.

A foreign key is vital for defining relationships in the database. It can be specified in the IBExpert Table Editor (started from the DB Explorer) on the Constraints page.



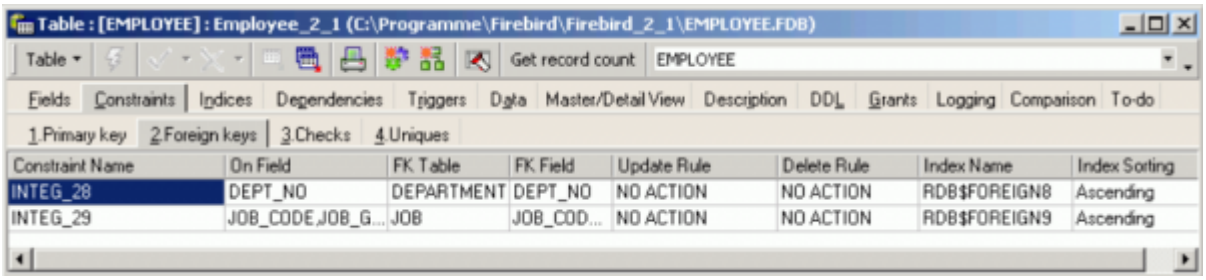
Foreign keys are used mainly for so-called reference tables. In a table storing, for example, employees, it needs to be determined which department each employee belongs to. Possible entries for the department number of each EMPLOYEE data set are contained in the DEPARTMENT table. As the EMPLOYEE table refers to the DEPT_NO as the primary key for the DEPARTMENT table, there is a foreign key relationship between the EMPLOYEE table and the DEPARTMENT table. Foreign key relationships are automatically checked in Firebird/InterBase®, and data sets with a non-existent department number cannot be saved.

When a primary key:foreign key relationship links to a single row in another table, what is known as a virtual row is created. The columns in that second table provide additional description about the primary key of the first table. This is also known as a 1:1 relationship.

A foreign key can also point to itself. Firebird enables you to reference recursive data and even represent tree structures in this way.

Foreign keys and their system names can be defined and viewed on the IBExpert Table Editor / Constraints page.

Since version 1.5 Firebird allows a USING INDEX subclause to be placed at the end of a primary, unique or foreign key definition. Please refer to the *Firebird 2.0 Language Reference Update* chapter, USING INDEX subclause for further information.



A primary key does not have to reference a foreign key. However a unique index is insufficient; a unique constraint needs to be defined (this definition also causes a unique index to be automatically generated).

When defining a foreign key, it is necessary to specify update and delete rules. Please refer to Referential integrity and Cascading referential integrity for further information.

SQL syntax:

```
ALTER TABLE MASTER
ADD CONSTRAINT UNQ_MASTER UNIQUE (FIELD_FOR_FK);
```

Foreign key names are limited to 32 characters up until InterBase® 6 and Firebird 1.5; InterBase® 7 allows 64 characters. IBExpert therefore recommends limiting table names to 14 characters, so that the foreign key name can include both related table names: prefix FK plus two separators plus both table names, e.g. FK_Table1_Table2.

Please note however that this is not an Firebird/InterBase® restriction, but purely an IBExpert recommendation to enable a clear and logical naming convention for foreign keys.

Note: if data has already been input in a table which is to subsequently be assigned a foreign key, this will not be allowed by Firebird/InterBase®, as it violates the principle of referential integrity. It is however possible to filter and delete the old data (where no reference to a primary key has been made) using a SELECT statement and committing. It is important to then disconnect and reconnect the database in IBExpert, for this to work.

New to Firebird 2.0: Creating foreign key constraints no longer requires exclusive access - Now it is possible to create foreign key constraints without needing to get an exclusive lock on the whole database.

Should you wish to delete constraints defined for a unique, foreign or secondary key, use the IBExpert Table Editor. Alternatively you can find a list of all constraints specified in a database in the system table, RDB$RELATION_NAME.

back to top of page

Candidate key

Any column or group of columns which can uniquely identify a data set, and can therefore be considered for use as a primary key. It is always NOT NULL (i.e. must not be left undefined), and unique, which means that the values in theses column(s) must never change. It is therefore

unadvisable to use columns such as surnames, telephone numbers and the like. Candidate keys do not have to be used as primary or foreign keys, and the automatically generated primary key ID remains the preferred method.

Alternate key/secondary key

In addition to primary keys, it is sometimes desirable to define *alternate* or *secondary* keys, for example, in a project table where, in addition to the primary key ID field, you wish to ensure that each project name is only used once.

back to top of page

Simple key

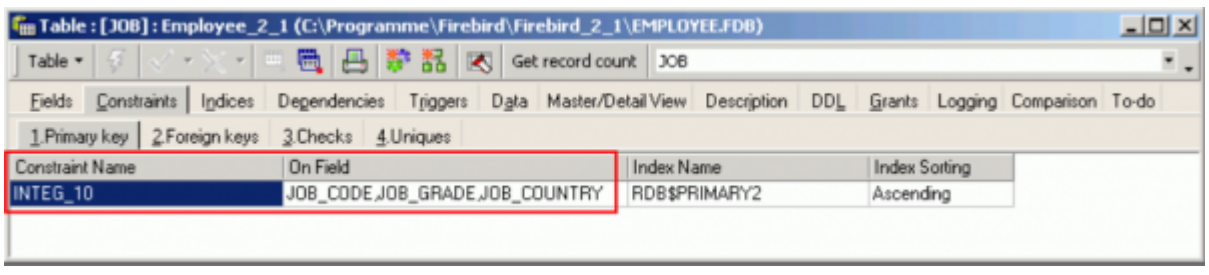
A simple key is composed of one column only, i.e. a single column is designated as a table's primary key.



back to top of page

Composite key/compound key

A composite key consists of two or more columns, designated together as a table's primary key. Multiple-column primary keys can be defined only as table-level constraints:



Single-column primary keys can be defined at either the column or the table level (but not both). For example, the following code states that the table's primary key consists of three columns, JOB_CODE, JOB_GRADE, and JOB_COUNTRY. Neither of these columns is required to be unique by itself, but their

combined value must be unique (and NOT NULL).

```
CREATE TABLE
COLUMN_defs ...
PRIMARY KEY (JOB_CODE,JOB_GRADE,JOB_COUNTRY);
```

Unfortunately such keys have two huge disadvantages: firstly they slow the database performance considerably, as Firebird/InterBase® needs to check all contents of all columns designated in such a composite key; secondly the sequence of the fields concerned must be identical in all referenced tables.

Basically composite keys should be avoided! It is much preferable to use an internal ID key (so-called artificial key) as the primary key for each table.

back to top of page

Unique

Unique fields are unequivocal, unambiguous, one-of-a-kind (i.e. there is no duplicate information allowed in the data sets of a unique field). Such fields must therefore also be NOT NULL.

Unique fields are given a unique index. Each unique field is a candidate (secondary) key.

back to top of page

Artficial key/surrogate key/alias key

An artificial or alias or surrogate key is created by the database designer/developer if there is no candidate key, i.e. no logical, simple field to be the primary key. An artificial key is a short ID number used to uniquely identify a record.

Such an internal primary key ID is recommended for all tables. They should always be invisible to the user, to prevent any potential external influence regarding their appearance and composition.

It is always wise to keep the primary key as short as possible to minimize the amount of disk space required, and to improve performance; therefore artificial keys should also be as short as possible. An ideal solution for the generation of an artificial key is the use of an autoincrement generator ID number.

IBExpert recommends this solution be used as an internal primary key for all tables.

Usually such an artificial/alias/surrogate key is just an autoincrement integer field so that each record has it's own unique integer identifier. For example:

```
CREATE TABLE CUSTOMERS (
   CUSTOMER_ID INTEGER NOT NULL,
   FIRST_NAME VARCHAR(20),
```

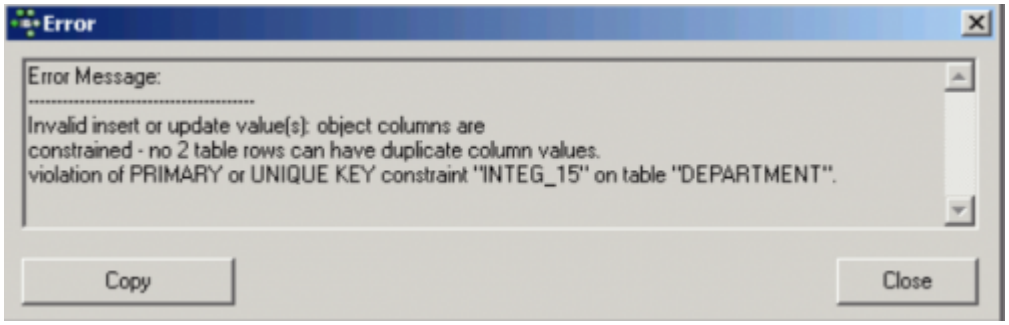
```
   MIDDLE.NAME VARCHAR(20),
   LAST_NAME VARCHAR(20);
...);
```

In this case CUSTOMER_ID the artificial or surrogate key.

back to top of page

Key violation

When a data set is created or changed, Firebird/InterBase® immediately checks the validity of the primary key. If the number already exists, or the field has been left blank, a key violation results, and the storage process is immediately cancelled.



Firebird/InterBase® immediately sends an error message referring to the violation of a unique or primary key constraint.

back to top of page

Referential integrity

The relationship between a foreign key and its referenced primary key is the mechanism for maintaining data consistency and integrity. Referential integrity ensures data integrity between tables connected by foreign keys. A foreign key is one or more columns that reference a primary key, i.e. when a value is entered in the foreign key, Firebird/InterBase® checks that this value also exists in the referenced primary key, so maintaining referential integrity.



Referential integrity can occur in the following three cases:

1. In the master table a data set is deleted. For example, the deletion of a customer, for whom there are still existing orders could lead to order data sets without a valid customer number. This could falsify analyses and lists, as the internal relationships no longer appear. The prevention of data set deletion in the master table, when data sets still exist in the detail table, is called prohibited deletion. The relay of deletions to all detail tables is called cascading deletion.
2. The primary key is changed in the master table. For example a customer is given a new customer number, so that all orders relating to this customer need to also relate to the new customer number. This is known as a cascading update.
3. A new data set is created, and the foreign key does not exist in the master table. For example an order is input with a customer number not yet allocated in the master table. A possible solution could be the automatic generation of a new customer. This is called a cascading insert.

Referential integrity is supported natively in Firebird/InterBase®, i.e. all foreign key basic relationships are automatically taken into consideration during data alterations. Since Version 5, InterBase® supports declarative referential integrity with cascading deletes and updates. In older versions, this could be implemented with triggers.

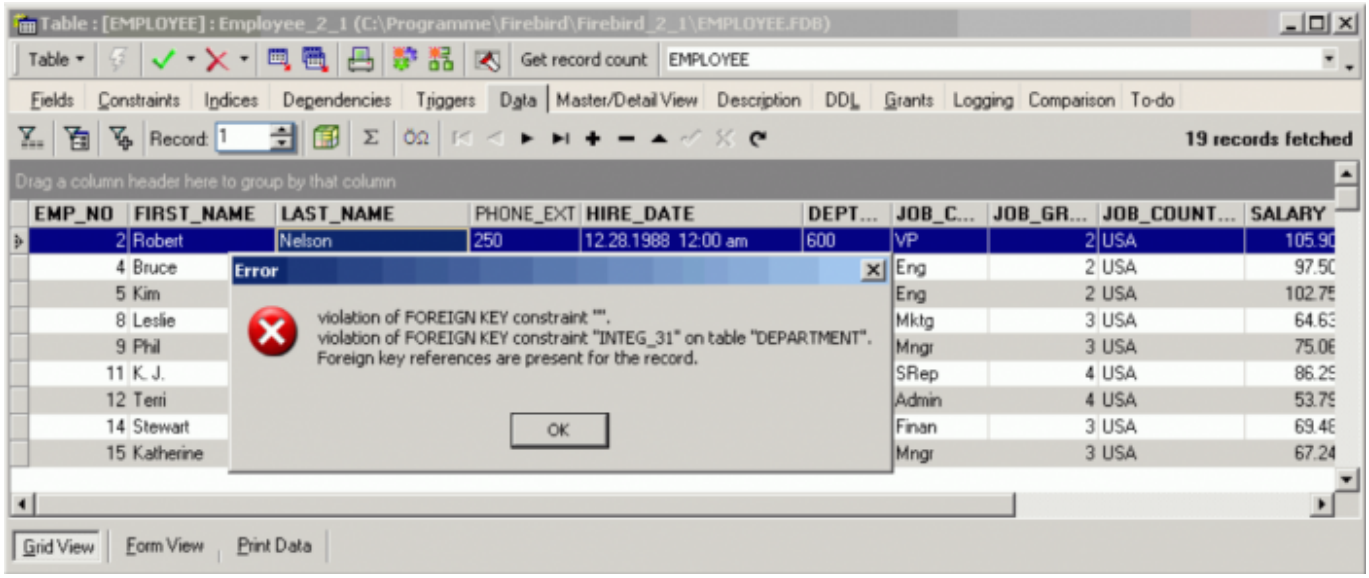
back to top of page

Cascading referential integrity

Since InterBase® v5/Firebird, cascading referential integrity is also supported.

When a foreign key relationship is specified, the user can define which action should be taken following changes to, or deletion of its referenced primary key. ON UPDATE defines what happens when the primary key changes and ON DELETE specifies the action to be taken when the referenced primary key is deleted. In both cases the following options are available:

1. **NO ACTION:** throws an exception if there is a existing relationship somewhere in another table:



2. **CASCADE:** the foreign key column is set to the new primary key value. A very handy function when

it comes to updating, as all referenced foreign key fields are automatically updated. When deleting the CASCADE option also deletes the foreign key row when the primary key is deleted. Be extremely careful when using CASCADE ON DELETE; when you delete a customer, you delete his orders, order lines, address, everything where there is a defined key relationship. It is safer to write a procedure that ensures just those data sets necessary are deleted in the right order. 3. **SET NULL:** if the foreign key value is allowed to be NULL, when a primary key value is deleted, it will set the relevant foreign key fields referencing this primary key value also to NULL. 4. **SET DEFAULT:** the foreign key column is set to its default value when a primary key field is deleted.

From:
https://ibexpert.com/docu/ - **IBExpert**

Permanent link:
https://ibexpert.com/docu/doku.php?id=01-documentation:01-13-miscellaneous:glossary:key

Last update: **2023/08/17 19:40**