Unique key

From Wikipedia, the free encyclopedia

In database relational modeling and implementation, a **unique key** is a set of zero, one, or more attributes. The value(s) of these attributes are required to be unique for each tuple (row) in a relation. The value, or combination of values, of **unique key** attributes for any tuple should not be repeated for any other tuple in that relation.

When more than one column is combined to form a unique key, their combined value is used to access each row and maintain uniqueness. These keys are referred to as aggregate. or compound keys. Values are not combined, they are compared using their data types.

When a column or set of columns is defined as unique to the database management system, the system verifies that each set of value(s) is unique before assigning the constraint. After the column(s) are defined as unique, an error will occur if an insertion is attempted with values that already exist. Some systems do not allow key values to be updated, and all systems will not allow duplicates. This ensures that uniqueness is maintained in both the primary table and any relations that are later bound to it.

Contents

- 1 Summary
- 2 Defining primary keys in SQL
- 3 Defining other keys in SQL
- 4 Surrogate keys
- 5 Alternate key
- 6 See also
- 7 References
- 8 External links

Summary

Keys provide the means for database users and application software to identify, access and update information in a database table. There may be several keys in any given table. For example, two distinct keys in a table of employees might be Badge Number and Login Name. The enforcement of a key constraint (i.e. a uniqueness constraint) in a table is also a data integrity feature of the database. The DBMS prevents updates that would cause duplicate key values and thereby ensures that tables always comply with the desired rules for uniqueness. Proper selection of keys when designing a database is therefore an important aspect of database usability and accuracy.

A relational database table by definition must have one or more keys (formally called candidate keys). By convention one of those keys per table is designated the "primary" key. The remaining (non-primary) keys are called alternate keys or sometimes secondary keys. Although mainly used today in the relational database context, the terms primary key and secondary key pre-date the relational model and are also used in other database models.

In relational database terms, a primary key need not differ in form or function from a key that isn't primary and in practice various different motivations may determine the choice of any one key as primary over another. The designation of a primary key may indicate the "preferred" identifier for data in the table, or that the primary key is to be used for foreign key references from other tables or it may indicate some other technical rather than semantic feature of the table. Some languages and software have special syntax features that can be used to identify a primary key as such (e.g. the PRIMARY KEY constraint in SQL).

Any key may consist of zero, one or more attributes. For example, a Social Security Number might be a single attribute key for an employee; a combination of flight number and date might be a key consisting of two attributes for a scheduled flight.

There are several types of unique keys used in database modeling and implementations.[1]

Key Name	Definition
Simple	A key made from only one attribute.
Concatenated	A key made from more than one attribute joined together as a single key, such as part or whole name with a system generated number appended as often used for E-mail addresses.
Compound	A key made from at least two attributes or simple keys, only simple keys exist in a compound key.
Composite	A key containing at least one compound key with at least one other attribute or simple key (this is an extension of a compound key).
Natural	A key made from data that exists outside the current database. In other words, the data is not system generated, such as a social security number imported from another system.
Surrogate	An artificial key made from data that is system assigned or generated when another candidate key exists. Surrogate keys are usually numeric ID values and often used for performance reasons.
Candidate	A key that may become the primary key.
Primary	The key that is selected as the primary key. Only one key within an entity is selected to be the primary key. This is the key that is allowed to migrate to other entities to define the relationships that exist among the entities. When the data model is instantiated into a physical database, it is the key that the system uses the most when accessing the table, or joining the tables together when selecting data.
Alternate	A non-primary key that can be used to identify only one row in a table. Alternate keys may be used like a primary key in a single-table select.
Foreign	A unique key that has migrated to another entity.

At the most basic definition, "a key is a unique identifier",[2] so *unique* key is redundant. Keys that are within their originating entity are unique within that entity. Keys that migrate to another entity may or may not be unique, depending on the design and how they are used in the other table. Foreign keys may be the primary key in another table; for example a PersonID may become the EmployeeID in the Employee table. In this case, the EmployeeID is both a foreign key and the unique primary key, meaning that the tables have a 1:1 relationship. In the case where the person entity contained the biological father ID, the father ID would not be expected to be unique because a father may have more than one child.

Here is an example of a primary key becoming a foreign key on a related table. ID migrates from the Author table to the Book table.

```
Author Table Schema:

Author(ID, Name, Address, Born)

Book Table Schema:

Book(ISBN, AuthorID, Title, Publisher, Price)
```

Here ID serves as the primary key in the table 'Author', but also as AuthorID serves as a Foreign Key in the table 'Book'. The Foreign Key serves as the link, and therefore the connection, between the two related tables in this sample database.

In a relational database, a candidate key uniquely identifies each row of data values in a database table. A candidate key comprises a single column or a set of columns in a single database table. No two distinct rows or data records in a database table can have the same data value (or combination of data values) in those candidate key columns since NULL values are not used. Depending on its design, a database table may have many candidate keys but at most one candidate key may be distinguished as the primary key.

A key constraint applies to the set of tuples in a table at any given point in time. A key is not necessarily a unique identifier across the population of all *possible* instances of tuples that could be stored in a table but it does imply a data integrity rule that duplicates should not be allowed in the database table. Some possible examples of unique keys are Social Security Numbers, ISBNs, vehicle registration numbers or user login names.

The relational model, as expressed through relational calculus and relational algebra, does not distinguish between primary keys and other kinds of keys. Primary keys were added to the SQL standard mainly as a convenience to the application programmer.

Unique keys as well as primary keys may be logically referenced by foreign keys, but most RDBMS only allow a foreign key constraint against a primary key.

Defining primary keys in SQL

Primary keys are defined in the ANSI SQL Standard, through the PRIMARY KEY constraint. The syntax to add such a constraint to an existing table is defined in SQL:2003 like this:

```
 ALTER TABLE <table identifier>
     ADD [ CONSTRAINT <constraint identifier> ]
     PRIMARY KEY ( <column expression> {, <column expression>}... )
```

The primary key can also be specified directly during table creation. In the SQL Standard, primary keys may consist of one or multiple columns. Each column participating in the primary key is implicitly defined as NOT NULL. Note that some RDBMS require explicitly marking primary key columns as `NOT NULL`.

```
CREATE TABLE table_name (

   ...
)
```

If the primary key consists only of a single column, the column can be marked as such using the following syntax:

```
CREATE TABLE table_name (
   id_col  INT  PRIMARY KEY,
   col2    CHARACTER VARYING(20),
   ...
)
```

Differences between Primary Key constraint and Unique constraint:

Primary Key constraint
1. A primary key *cannot* allow null (a primary key cannot be defined on columns that allow nulls).
2. Each table cannot have more than one primary key.
3. On some RDBMS a primary key generates a clustered index by default.

Unique constraint
1. A unique constraint can be defined on columns that allow nulls.
2. Each table can have multiple unique keys.
3. On some RDBMS a unique key generates a nonclustered index by default.

Defining other keys in SQL

The definition of other unique keys is syntactically very similar to primary keys.

```
ALTER TABLE <table identifier>
    ADD [ CONSTRAINT <constraint identifier> ]
    UNIQUE ( <column expression> {, <column expression>}... )
```

Likewise, unique keys can be defined as part of the `CREATE TABLE` SQL statement.

```
CREATE TABLE table_name (
   id_col   INT,
   col2     CHARACTER VARYING(20),
   key_col  SMALLINT NOT NULL,
   ...
   CONSTRAINT key_unique UNIQUE(key_col),
   ...
)
```

```
CREATE TABLE table_name (
    id_col  INT  PRIMARY KEY,
    col2    CHARACTER VARYING(20),
    ...
    key_col  SMALLINT NOT NULL UNIQUE,
    ...
)
```

Note that unlike the PRIMARY KEY constraint a UNIQUE constraint does not imply NOT NULL for the columns participating in the constraint. NOT NULL must be specified to make the column(s) a key. It is possible to put UNIQUE constraints on nullable columns but the SQL standard states that the constraint does not guarantee uniqueness of nullable columns (uniqueness is not enforced for rows where any of the columns contains a null).

According to the SQL[3] standard a unique constraint does not enforce uniqueness in the presence of nulls and can therefore contain several rows with identical combinations of nulls and non-null values — however not all RDBMS implement this feature according to the SQL standard.[4][5]

Surrogate keys

In some circumstances the natural key that uniquely identifies a tuple in a relation may be cumbersome to use for software development. For example, it may involve multiple columns or large text fields. In such cases, a surrogate key can be used instead as the primary key. In other situations there may be more than one candidate key for a relation, and no candidate key is obviously preferred. A surrogate key may be used as the primary key to avoid giving one candidate key artificial primacy over the others.

Since primary keys exist primarily as a convenience to the programmer, surrogate primary keys are often used, in many cases exclusively, in database application design.

Due to the popularity of surrogate primary keys, many developers and in some cases even theoreticians have come to regard surrogate primary keys as an inalienable part of the relational data model. This is largely due to a migration of principles from the Object-Oriented Programming model to the relational model, creating the hybrid object-relational model. In the ORM, these additional restrictions are placed on primary keys:

- Primary keys should be immutable, that is, never changed or re-used; they

should be deleted along with the associated record.
- Primary keys should be anonymous integer or numeric identifiers.

However, neither of these restrictions is part of the relational model or any SQL standard. Due diligence should be applied when deciding on the immutability of primary key values during database and application design. Some database systems even imply that values in primary key columns cannot be changed using the `UPDATE` SQL statement.

Alternate key

Typically, one candidate key is chosen as the primary key. Other candidate keys become alternate keys, each of which may have a unique index assigned to it in order to prevent duplicates (a duplicate entry is not valid in a unique column).[6]

Alternate keys may be used like the primary key when doing a single-table select or when filtering in a *where* clause, but are not typically used to join multiple tables.

See also

- Globally Unique Identifier
- Natural key
- Persistent Object Identifier

References

1. "Choosing a Primary Key: Natural or Surrogate?". *AgileData.org*. Retrieved August 28, 2014.
2. Awad, Elias (1985), *Systems Analysis and Design, Second Edition*, Richard D. Irwin, Inc., ISBN 0-256-02824-9
3. Summary of ANSI/ISO/IEC SQL (http://www.xcdsql.org /Summary%20of%20SQL.html#chapter-Table%20constraints)
4. Constraints - SQL Database Reference Material - Learn sql, read an sql manual, follow an sql tutorial, or learn how to structure an SQL query (http://www.sql.org /sql-database/postgresql/manual/ddl-constraints.html#AEN1832)
5. Comparison of different SQL implementations (http://troels.arvin.dk/db/rdbms /#constraints-unique)
6. Alternate key - Oracle FAQ (http://www.orafaq.com/wiki/Alternate_key)

External links

- Relation Database terms of reference, Keys (http://rdbms.opengrass.net /2_Database%20Design/2.1_TermsOfReference/2.1.2_Keys.html): An overview of the different types of keys in an RDBMS

Categories: Database management systems | Data modeling