|  |  |  |
| --- | --- | --- |
|  | Summary | Description |
|  | Missing Primary Keys | A primary key uniquely identifies rows in tables. Missing a primary/ unique key on a table allows duplication of rows, which should be avoided. Furthermore, individual rows cannot be referenced using foreign keys when the table lacks a primary/ unique key. If a table does not contain columns suitable for a primary key, it is always possible to create a surrogate key. |
|  | Different Data Type Between Source and Target Columns in a Foreign Key | A foreign key is a relationship between two tables, a source table and a target table. Values from the source column is stored in the target column, hence the data type of the two columns should be the same. However, it is possible to create a working foreign-key relationship between two columns of different data types. For instance, a source column "number(8)" and a target column "number(4)". This may lead to an application crash when inserting data because the domain of the target column is smaller than the source column. |
|  | Varchar Columns of Length Zero | A column designed to contain no data is simply a bad design practice. A varchar of length 0 could be used to represent boolean values, such that the empty string equals true and a null value equal false. However, there are better and less obscure ways to model boolean values. |
|  | Inconsistent Naming Convention | Using consistent naming of columns and tables makes life easier for the database designer and application programmers. An inconsistent naming convention complicates writing queries and understanding the schema. |
|  | Inappropriate Length of Default Value For Char Columns | A char column always occupies the specified length, even when the empty string is used. Therefore, char columns should only be used if the length is small or the size of the data is known in advance. Otherwise, varchar columns should be used because they occupy only the space corresponding to the actual data. |
|  | Redundant Foreign Keys | Duplicate foreign keys could have contradicting referential actions, such as "CASCADE" and "SET NULL". Having contradicting referential actions may lead to unforeseen events when, e.g., deleting rows. Furthermore, if the foreign-keys have indices the DBMS will have to maintain more indices. A duplicate foreign key can be deleted with little effort. |
|  | Table With Too Few Columns | Tables with zero or one column are suspicious. A table with zero columns cannot contain any data. A table with one column can be accepted under special circumstances, but should generally be avoided. |
|  | Too Big Indices | Large indices reduce performance because they are expensive to maintain, and should be avoided when smaller keys are sufficient. Some DBMSs have a maximum key size on indices, e.g., SQL Server is limited to 900 bytes per key. In some cases, a large natural primary key can be replaced with a surrogate key. |
|  | Too Many Nullable Columns | In DBLint there are two cases where a table is said to contain too many nullable columns:   * All columns are nullable except the primary key columns. * A large percentage of the columns are nullable.   The first case is especially bad if the primary key is a single surrogate key, because a row can contain no useful data. In the second case, it is likely that the developer forgot to add the appropriate not-null constraints. |
|  | Too Long Column Names | The maintainability of a schema might decrease with long names, because it makes identifiers harder to remember and queries more difficult to write. Furthermore, Oracle does not allow column names to exceed 30 characters. |
|  | Nullable and Unique Columns | Null in a database typically refers to "value does not exist" or "value unknown", and as such should not be allowed in columns, which have a unique constraint defined. Null values in unique indices are handled differently depending on DBMS: Some DBMSs allows zero or one null value in a unique index, while, e.g., Oracle allows multiple null values. This difference may be a portability issue, and cause misunderstandings among developers of the different DBMSs. |
|  | Cycles Between Tables | A cycle can be necessary to model specific data structures, e.g., a hierarchical structure. However, the developer should be aware that the cycle exists, because circular dependencies may cause several problems if deferrability and delete rules are not considered. These problems are the following.   * If there is a cascade delete on all references, it is possible to delete all data in the tables. * If no references are deferred and the columns are mandatory, data cannot be inserted. |
|  | Inconsistent Max Lengths of Varchar Columns | Inconsistent maximum length of varchar columns is a rule purely about consistency. Consider an example with 200 columns of maximum length 256 and three columns of length 255. These three columns are deviating from the majority, and could be 256 without conflicting with the data in the columns. |
|  | Self-Referencing Primary Key | Having a foreign key relation on a primary key column referencing itself strongly indicates an error. The foreign key must reference its own row and does not contain any useful information. Such a foreign key can be deleted without any loss of functionality or conflicts in the database. |
|  | Inconsistent Data Types in Column Sequence | A sequence of related columns can be inferred from the naming, e.g. ("address\_1", "address\_2", ..., "address\_n"). Another example is columns used for extensibility, e.g., 10 columns ("cust\_col\_1", ..., "cust\_col\_10"), used to store unforeseen information after the database is deployed.  All columns in the sequence should have the same data type to avoid confusion and potential errors. Imagine that there are 10 columns in a sequence and the third column's data type is integer and the others are varchars. This may result in problems because a developer might mistake the third column for being a varchar, like the others. Furthermore, varying data types in a column sequence violates consistency. |
|  | Missing Column in a Sequence of Columns | If there exist a sequence of columns, e.g., ("col\_1", "col\_2", ..., "col\_n"), the postfix number should be ordered sequential from 1 to n. If a column is missing from a sequence it has probably been forgotten or deleted without proper refactoring. |
|  | Primary- and Unique-Key Constraints on the Same Columns | Having a primary- and unique-key constraint on the same columns makes the unique constraint redundant. The unique key can be deleted without affecting data integrity. |
|  | Redundant Indices | Redundant indices are usually not necessary. A redundant index is an index where the sequence of columns is a prefix of another index, e.g., the index "inx\_a(col\_1)" is redundant to "inx\_b(col\_1, col\_2)". Having redundant indices is a performance issue because the DBMS needs to maintain more data structures than necessary. There are exceptions where a redundant index is reasonable, but most likely it can be deleted without any problems. |
|  | Too Short Column Names | Columns should be named with meaningful and distinct names. This makes it easy to read and understand the data model and queries. Very short column names have a tendency to consist of abbreviations or letters that have certain meaning in the development team. However, these columns are not very maintainable and make queries less understandable. |
|  | Too Many Text Columns in a Table | LOB columns containing text are used to store large string values. Normally they will only take up the space they need, however the data are stored outside the table, and hence it requires an additional I/O for each value. If a table contains a large number of these columns it could indicate that the developer were unaware of the different data types. |
|  | Foreign-Key Without Index | When deleting/updating a row from the referenced table, the DBMS checks that the specific row is not referenced, and takes corresponding action depending on the delete/update rule. This check must look-up values in the referencing table, which requires a full table scan if an index does not exist. Having an index on the foreign-key columns will make this look-up faster. |
|  | Primary-Key Columns Not Positioned First | It is convention to position the primary-key columns first. The order of columns in a table is important for readability purposes. A related case is when a table contains a sequence of columns, such as ("address\_1", "address\_2", ...), and it is natural to place the columns ascending based on the postfix number. Similarly placing the primary-key columns first makes it possible to quickly see how rows are uniquely identified. |
|  | Use of Reserved Words From SQL | Reserved SQL keywords such as "date" and "from" should be avoided when choosing identifiers. Avoiding reserved SQL keywords in identifiers makes the queries more readable and names will not need to be escaped in queries. |
|  | Different Data Types for Columns With the Same Name | A column's name often refers to a concept, hence when the same name is used with different data types the representation of that concept is inconsistent. Possible errors that could arise include implicit casts.  SQL clauses such as natural join and using, matches columns based on names. Without care two columns could easily be matched, which will make implicit casts.  Generic names such as "value" and "content" do not necessarily refer to the same concepts. |
|  | Use of Special Characters in Identifiers | Special characters in identifier names should be avoided, except the character "\_" for the following two reasons.   * Identifiers must be escaped in queries. * Identifiers cannot be mapped directly into programming languages.   In practice there are almost no good reasons for using special characters instead of an understandable/describing name. For instance, a product table containing a column with products numbers, could be named "#" but a better solution would simply be "product\_no". |
|  | Table Islands | Having a connected schema graph means that the data is related. If the schema is not connected it is possible that one is trying to model two separate concepts or businesses. In that case it is better to extract the table islands into separate schemas. |
|  | Too Large Varchar Columns | Large varchar columns are a problem because they may cause the row to overflow resulting in chaining. Chained rows are slower to extract from the database as they require additional I/Os. |