|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Summary | Description | Linter | Needs Configuration Parameter | Table Level? |
|  | Missing Primary Keys | A primary key uniquely identifies rows in tables. It allows individual rows to be referenced in queries, and foreign keys. Good database designs use surrogate key as the primary key. |  |  |  |
|  | Different Data Type Between Primary and Foreign Keys | 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. Some database designs use VARCHAR(0) for Boolean values, but this is not preferred design. |  |  |  |
|  | 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. |  |  |  |
|  | 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 should be avoided. | LinterTableWithSingleColumn |  |  |
|  | 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 | All columns are nullable except the primary key columns.   * A row can contain no useful data. |  |  |  |
|  | Nullable and Unique Columns | Null values should not be allowed in columns which have a unique constraint defined. | LinterTableWithNullColumnsInIndex |  |  |
|  | Cycles Between Tables | Cyclical relationships between tables can potentially cause issues with deletes and inserts. |  |  |  |
|  | 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. |  |  |  |
|  | Missing Column in Consecutively Numbered Columns | If a column is missing from a consecutively numbered columns, it has probably been forgotten or deleted without proper refactoring. |  |  |  |
|  | Inconsistent Data Types in Consecutively Numbered Columns | All columns in the sequence should have the same data type to avoid confusion and potential errors. |  |  |  |
|  | Missing Column in Consecutively Numbered Columns | If a column is missing from consecutively numbered columns, it has probably been forgotten or deleted without proper refactoring. |  |  |  |
|  | Redundant Indices | 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)". |  |  |  |
|  | Too Many LOB Columns in a Table | LOB columns containing text are used to store large string values. Typically the data is stored outside the table, and needs additional reads or writes for each value. |  |  |  |
|  | 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". |  |  |  |
|  | 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. |  |  |  |