**Naming Conventions - Database for IST 261**

***/\**** *Based and derived from Simon Holywell’s SQL Style Guide:* [*http://www.sqlstyle.guide/*](http://www.sqlstyle.guide/)

*SQL Style Guide is licensed under a* [*Creative Commons Attribution-ShareAlike 4.0 International License.*](http://creativecommons.org/licenses/by-sa/4.0/) ***\*/***

### **General**

* Ensure the name is unique and does not exist as a reserved keyword.
* Keep the length to a maximum of 30 bytes — in practice these is 30 characters unless you are using multi-byte character set.
* Names must begin with a letter and may not end with an underscore.
* Only use letters and numbers in names.
* Avoid the use of underscores wherever possible.
* Make use of CamelCase with no spaces/no underscores during naming. Example: (“first name” becomes “FirstName”).
* Avoid abbreviations and if you have to use them make sure they are commonly understood.

### **Tables**

* Table names should be capitalized (to easily differentiate between tables and columns at a glance). For example, TABLENAME.
* Use a collective name or, less ideally, a plural form. For example (in order of preference) STAFF and EMPLOYEES.
* Do not prefix with tbl or any other such descriptive prefix.
* Never give a table the same name as one of its columns and vice versa.
* Make use of TABLEX\_TABLEY for linked tables, except for a situation where renaming would make the linked table easier to understand. For example: concatenating two table names together to create the name of a relationship table. Rather than STUDENT\_PERFORMANCE, make use of GRADES.

### **Columns**

* Always use the singular name (e.g., FirstName).
* Where possible avoid simply using id as the primary identifier for the table.
* Do not add a column with the same name as its table and vice versa.
* Always use lowercase except where it may make sense not to such as proper nouns.
* Foreign keys should be named with their parent table and the primary key’s name to make it clear what they are and what they reference (e.g., parent STUDENT, primary key ID, STUDENT\_ID is the foreign key that references STUDENT (ID) ).
* Make use of the standard SQL datatypes (refer next section) as the prefix to the column name. For example, instead of “WebUrl TEXT” make use of “varchar\_WebUrl TEXT” or “vrchrWebUrl TEXT”. This is because there might be the possibility of tables getting imported from/exported to other database types in the future.

### **Standard SQL datatypes**

### /\* Fetched from: <https://www.w3schools.com/SQl/sql_datatypes_general.asp> \*/

|  |  |
| --- | --- |
| **Data type** | **Description** |
| CHARACTER(n) | Character string. Fixed-length n |
| VARCHAR(n) or  CHARACTER VARYING(n) | Character string. Variable length. Maximum length n |
| BINARY(n) | Binary string. Fixed-length n |
| BOOLEAN | Stores TRUE or FALSE values |
| VARBINARY(n) or  BINARY VARYING(n) | Binary string. Variable length. Maximum length n |
| INTEGER(p) | Integer numerical (no decimal). Precision p |
| SMALLINT | Integer numerical (no decimal). Precision 5 |
| INTEGER | Integer numerical (no decimal). Precision 10 |
| BIGINT | Integer numerical (no decimal). Precision 19 |
| DECIMAL(p,s) | Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal |
| NUMERIC(p,s) | Exact numerical, precision p, scale s. (Same as DECIMAL) |
| FLOAT(p) | Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision |
| REAL | Approximate numerical, mantissa precision 7 |
| FLOAT | Approximate numerical, mantissa precision 16 |
| DOUBLE PRECISION | Approximate numerical, mantissa precision 16 |
| DATE | Stores year, month, and day values |
| TIME | Stores hour, minute, and second values |
| TIMESTAMP | Stores year, month, day, hour, minute, and second values |
| INTERVAL | Composed of a number of integer fields, representing a period of time, depending on the type of interval |
| ARRAY | A set-length and ordered collection of elements |
| MULTISET | A variable-length and unordered collection of elements |
| XML | Stores XML data |

### 

### **Stored procedures**

* The name must contain a verb.
* Do not prefix with sp\_ or any other such descriptive prefix or Hungarian notation.

### **Reserved words**

* Always use uppercase for the reserved keywords like SELECT and WHERE.
* It is best to avoid the abbreviated keywords and use the full length ones where available (prefer ABSOLUTE to ABS).
* Do not use database server specific keywords where an ANSI SQL keyword already exists performing the same function. This helps to make code more portable.
* List of reserved words in SQLite: <http://www.sqlite.org/lang_keywords.html>
* Not advised. BUT if you do want to use a reserved keyword as a name in SQLite, then you need to quote it and this can be done in the following ways:

|  |  |  |
| --- | --- | --- |
| **'keyword'** |  | A keyword in single quotes is a string literal. |
| **"keyword"** |  | A keyword in double-quotes is an identifier. |
| **[keyword]** |  | A keyword enclosed in square brackets is an identifier. This is not standard SQL. This quoting mechanism is used by MS Access and SQL Server and is included in SQLite for compatibility. |
| **`keyword`** |  | A keyword enclosed in grave accents (ASCII code 96) is an identifier. This is not standard SQL. This quoting mechanism is used by MySQL and is included in SQLite for compatibility. |

#### Reserved words in SQL to keep in mind for the future: <https://www.drupal.org/docs/develop/coding-standards/list-of-sql-reserved-words>

#### 

#### **Spaces**

Spaces should be used to line up the code so that the root keywords all end on the same character boundary. This forms a river down the middle making it easy for the reader’s eye to scan over the code and separate the keywords from the implementation detail.

Although not exhaustive always include spaces:

* before and after equals (=)
* after commas (,)
* surrounding apostrophes (') where not within parentheses or with a trailing comma or semicolon.

#### **Line spacing**

Always include newlines/vertical space:

* before AND or OR
* after semicolons to separate queries for easier reading
* after each keyword definition
* after a comma when separating multiple columns into logical groups
* to separate code into related sections, which helps to ease the readability of large chunks of code.

Keeping all the keywords aligned to the right hand side and the values left aligned creates a uniform gap down the middle of query. It makes it much easier to scan the query definition over quickly too.

### **Indentation**

To ensure that SQL is readable it is important that standards of indentation are followed:

#### **Joins**

Joins should be indented to the other side and grouped with a new line where necessary. Refer the example:

**Example:**

**SELECT** RIDERS.LastName

**FROM** RIDERS

**INNER JOIN** BIKES

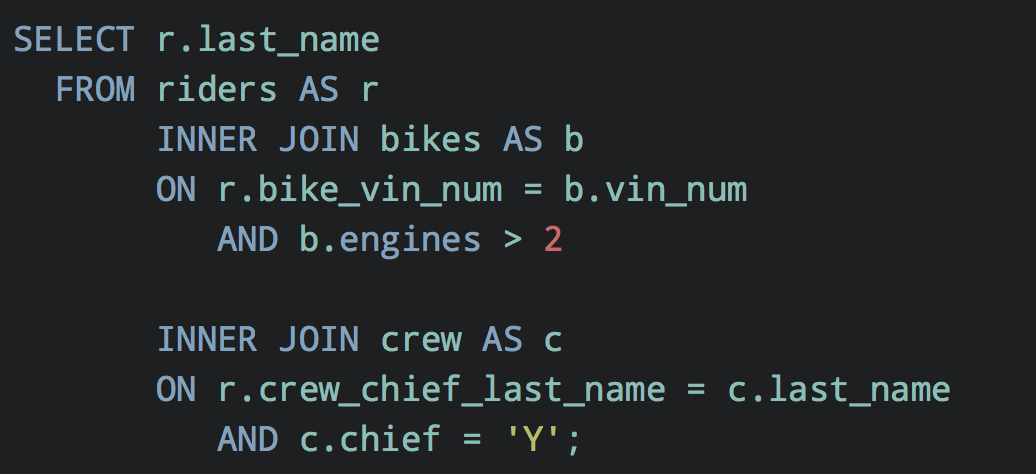
**ON** RIDERS.BikeVIN = BIKES.RIDERS\_BikeVIN

**AND** BIKES.Engines > 2

**INNER JOIN** CREW

**ON** RIDERS.CrewChiefLastName = CREW.LastName

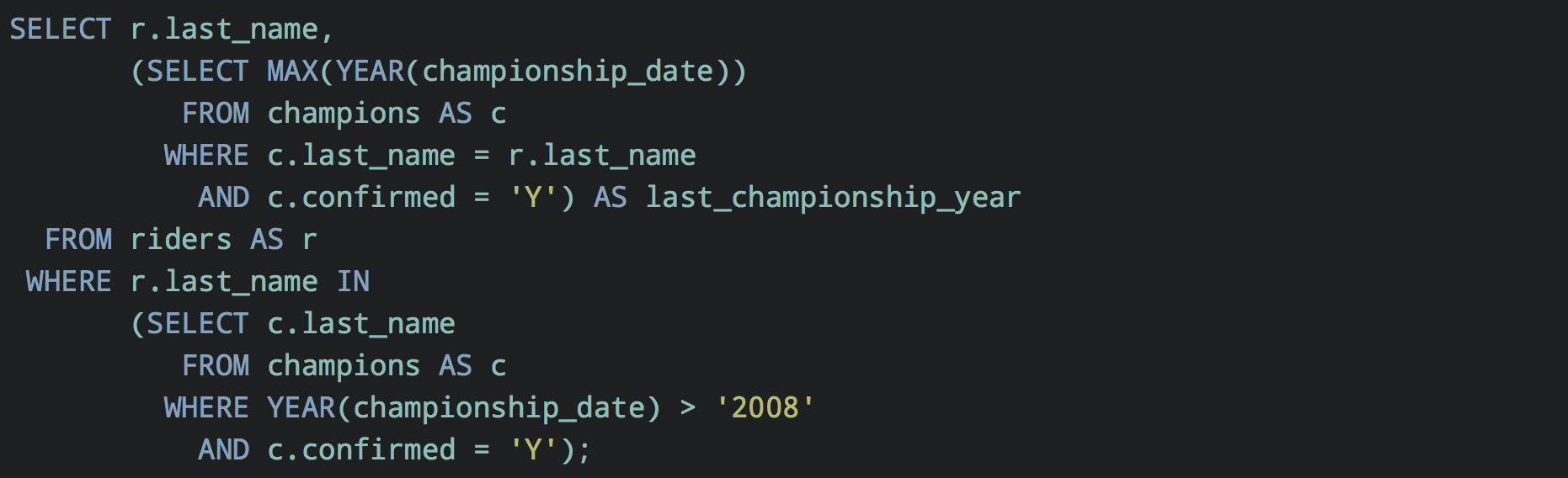
**AND** CREW.Chief = ‘Y’;



#### **Subqueries**

Subqueries should also be aligned to the right side of the river and then laid out using the same style as any other query. Sometimes it will make sense to have the closing parenthesis on a new line at the same character position as it’s opening partner—this is especially true where you have nested subqueries.

**Example:**



### **Preferred formalisms**

* Make use of BETWEEN where possible instead of combining multiple statements with AND.
* Similarly use IN() instead of multiple OR clauses.
* Where a value needs to be interpreted before leaving the database use the CASE expression. CASE statements can be nested to form more complex logical structures.
* Avoid the use of UNION clauses and temporary tables where possible. If the schema can be optimised to remove the reliance on these features then it most likely should be.

### **Constraints and keys**

Constraints and their subset, keys, are a very important component of any database definition. They can quickly become very difficult to read and reason about though so it is important that a standard set of guidelines are followed.

#### **Choosing keys**

Deciding the column(s) that will form the keys in the definition should be a carefully considered activity as it will effect performance and data integrity.

1. The key should be unique to some degree.
2. Consistency in terms of data type for the value across the schema and a lower likelihood of this changing in the future.
3. Keeping the key as simple as possible whilst not being scared to use compound keys where necessary.

It is a reasoned and considered balancing act to be performed at the definition of a database. Should requirements evolve in the future it is possible to make changes to the definitions to keep them up to date.

#### **Defining constraints**

Once the keys are decided it is possible to define them in the system using constraints along with field value validation.

##### **General**

* Tables must have at least one key to be complete and useful.
* Constraints should be given a custom name excepting UNIQUE, PRIMARY KEY and FOREIGN KEY where the database vendor will generally supply sufficiently intelligible names automatically.

##### **Layout and order**

* Specify the primary key first right after the CREATE TABLE statement.
* Constraints should be defined directly beneath the column they correspond to. Indent the constraint so that it aligns to the right of the column name.
* If it is a multi-column constraint then consider putting it as close to both column definitions as possible and where this is difficult as a last resort include them at the end of the CREATE TABLE definition.
* If it is a table level constraint that applies to the entire table then it should also appear at the end.
* Use alphabetical order where ON DELETE comes before ON UPDATE.
* If it make senses to do so align each aspect of the query on the same character position. For example all NOT NULL definitions could start at the same character position. This is not hard and fast, but it certainly makes the code much easier to scan and read.

##### **Validation**

* Use LIKE and SIMILAR TO constraints to ensure the integrity of strings where the format is known.
* Where the ultimate range of a numerical value is known it must be written as a range CHECK() to prevent incorrect values entering the database or the silent truncation of data too large to fit the column definition. In the least it should check that the value is greater than zero in most cases.
* CHECK() constraints should be kept in separate clauses to ease debugging.

**Example:**

