NULL

- SQL-Standard definiert NULL nicht als Wert, sondern als Platzhalter
- Bei der Oracle Datenbank kann ein leerer String NULL sein

NULL Indizieren

- Jeder Index ist ein partieller Index
- Zeilen werden in ein mehrspaltigen Index aufgenommen, wenn zumindest eine Indexspalte nicht NULL ist
- Funktionsbasierte Indexes
- Mythos: Oracle Datenbank kann nicht NULL indizieren

Anti-Patterns

- Anti-Patterns in SQL sind bewährte Praktiken oder Lösungsansätze, die vermieden werden sollten, da sie ineffizient, fehleranfällig oder schwerwartbar sind.
- Sie können zu schlechter Leistung, schwieriger Wartung oder sogar Datenproblemen führen

Date-Types

- Eine der häufigsten Verschleierung betrifft Datums-Spalten. Die Oracle Datenbank ist dafür besonders anfällig, weil sie nur den DATE-Typen hat, die immer eine Uhrzeit mitführen
- Um den Zeitanteil aus einer DATE-Spalte zu entfernen, hat sich die TRUNC-Funktion durchgesetzt

```
SELECT ...
  FROM sales
 WHERE TRUNC(sale_date) = TRUNC(sysdate - INTERVAL '1' DAY)
```

- Die Abfrage ist absolut korrekt, kann aber einen Index auf SALE_DATE nicht ordentlich nutzen
- Generische Lösung muss man die Bedingung als explizite Bereichsbedingung umformulieren

```
SELECT ...
  FROM sales
 WHERE sale_date BETWEEN quarter_begin(?)
                     AND quarter_end(?)
```

- Eine andere häufige Verschleierung ist, das Datum als Text zu vergleichen

```
SELECT ...
  FROM sales
 WHERE TO_CHAR(sale_date, 'YYYY-MM-DD') = '1970-01-01'
```

- Das Problem ist wieder die Konvertierung der Spalte SALE_DATE. Solche Bedingungen entstehen oft im Glauben, dass man einer Datenbank nur Zahlen und Texte übergeben kann.
- Mit Bind-Parametern kann man aber auch andere Daten-Typen verwenden
- Falls das nicht möglich ist, sollte man nicht die Tabellenspalte, sondern den Suchbegriff konvertieren

```
SELECT ...
  FROM sales
 WHERE sale_date = TO_DATE('1970-01-01', 'YYYY-MM-DD')
```

- Die folgende Verschleierung ist besonders tückisch:

```
sale_date LIKE SYSDATE
```

- Diese Bedingung scheint auf den ersten Blick nicht verschleiert zu sein, weil sie keine Funktion auf der Tabellenspalte verwendet.
- Durch die Verwendung des LIKE-Operators wird aber ein String Vergleich erzwungen
- Der Predicate-Information-Bereich des Ausführungsplanes zeigt, was die Oracle Datenbank macht
- ```
  filter( INTERNAL_FUNCTION(SALE_DATE)
     LIKE TO_CHAR(SYSDATE@!))
  ```

Numerische Strings

- Numerische Strings sind Zahlen, die in Text-Feldern gespeichert werden
- Es ist also dasselbe Problem wie zuvor. Durch die Funktion kann ein Index auf NUMERIC_STRING nicht sinnvoll genutzt werden
- Die Lösung ist wieder dieselbe: Anstatt den Spaltentypen an den Suchbegriff anzupassen, passt man den Suchbegriff an den Spaltentypen an
- ```
  SELECT ...
    FROM ...
   WHERE numeric_string = TO_CHAR(42)
  ```

Zusammenfügen von Spalten

- Wenn man vor diesem Problem steht, hat man aber nur in den seltensten Fällen die Möglichkeit die Tabelle umzubauen
- Manchmal muss man Bedingungen gezielt verschleiern, damit sie nicht als Zugriffsprädikat verwendet werden.
- ```
  SELECT last_name, first_name, employee_id
    FROM employees
   WHERE subsidiary_id = ?
     AND last_name LIKE ?
  ```
- Angenommen es gibt sowohl einen Index auf SUBSIDIARY_ID als auch auf LAST_NAME. Welcher ist für diese Abfrage besser?
- Ohne zu wissen, wo die Wildcard-Zeichen im Suchbegriff stehen, kann man keine qualifizierte Antwort geben
- Auch die Datenbank kann nur raten. Wenn man aber weiß, dass der Suchbegriff immer mit einem Wildcard Zeichen beginnt, kann man die entsprechende Bedingung absichtlich verschleiern.
- Dadurch kann der LIKE-Filter nicht als Zugriffsprädikat verwendet werden
- ```
  SELECT last_name, first_name, employee_id
    FROM employees
   WHERE subsidiary_id = ?
     AND last_name || '' LIKE ?
  ```
- Dafür genügt es, einen leeren String an die Spalte LAST_NAME anzuhängen. Dieses Vorgehen ist allerdings nur der letzte Ausweg, wenn es keine anderen Möglichkeiten mehr gibt

Schlaue Logiken

- SQL-Datenbanken unterstützen Ad-hoc-Abfragen, die zur Laufzeit analysiert und optimiert werden.
- Dynamische Abfragen sind also sinnvoll und können durch die Verwendung von Bind-Parametern effizient gestaltet werden.
- Es gibt jedoch eine verbreitete Praxis, bei der dynamische Abfragen durch statische ersetzt werden, aufgrund des Mythos, dass dynamisches SQL langsam ist.

- In Datenbanken mit zentralem Ausführungsplan-Cache kann diese Praxis jedoch mehr Probleme verursachen als sie löst
- Die richtige Lösung für dynamische Abfragen ist dynamisches SQL.
- Frei nach dem KISS-Prinzip sollte man in jeder Abfrage nur die relevanten Bedingungen anführen – sonst nichts

```
SELECT first_name, last_name, subsidiary_id, employee_id
  FROM employees
 WHERE UPPER(last_name) = :name
```

- Konstruktionen wie diese „schlaue“ Logik sind weiter verbreitet als man glaubt. Daher haben alle Datenbanken, Gegenmaßnahmen ergriffen – oft werden damit neue Probleme eingeführt

Mathematik

- Es gibt noch eine weitere Gruppe von Verschleierungen, die zwar schlau sind, aber die Indexnutzung unterbinden können. Anstatt logischer Ausdrücke verwendet man dabei Mathematik

```
SELECT numeric_number
  FROM table_name
 WHERE numeric_number - 1000 > ?
```

```
SELECT a, b
  FROM table_name
 WHERE 3*a + 5 = b
```

- Mit einem Funktions-basierten Index kann man all diese Beispiele indizieren

```
SELECT a, b
  FROM table_name
 WHERE 3*a - b = -5
```

- Für den linken Teil der Gleichung kann man dann einen Funktions-basierten Index anlegen

```
CREATE INDEX math ON table_name (3*a - b)
```

PAGGING

Drei Methoden:

1. Offset-Methode: am einfachsten aber nicht sehr schnell
2. Seek-Methode: schnell aber schwierig zu schreiben
3. Windows-Funktion: relativ einfach aber wird nicht von allen Datenbanken unterstützt