Teil 2:

Integrität im relationalen Modell, vom Relationalen Modell zum RDBMS, SQL Einführung

Integrität im relationalen Modell

Für eine 'relationale DB' im Sinne von Codd müssen gewisse Bedingungen immer erfüllt sein. Dazu wird im Folgenden der Grundbegriff des *Schlüssels* eingeführt.

Gefahr für die Integrität eines Datenbestandes lauert in Operationen wie *Einfügen*, *Löschen* oder *Ändern*.

Grundbegriffe:

Schlüssel: minimale Attributmenge zur eindeutigen Identifizierung eines Tupels in einer Relation

(z.B. Artikelnr, Kundennr, Lieferantennr).

Primärschlüssel (primary key, pkey): falls mehrere Schlüssel, der wichtigste

Alternativschlüssel (alternate key, secondary key, Sekundärschlüssel)

weitere eindeutige Identifizierungen (EANnr, SVNR)

pkey + altkey = *candidate keys*

Fremdschlüssel (foreign key): Attributmenge, die sich auf pkey einer anderen Relation bezieht (Referenz).

Schlüssel können 1 oder mehrstellig sein, d.h. aus 1 oder mehreren Attributen bestehen.

Primärschlüsselattribute werden im Schema unterstrichen dargestellt.

Beispiel

sch(Lieferant): {[<u>lifnr</u>, name, plz, ort, str, tel]}'
sch(Artikel): {[<u>artikelnr</u>, bezeichnung, preis, menge, lifnr]}

Primärschlüssel von Lieferant ist *lifnr*, von Artikel ist es *artikelnr*. *Artikel.lifnr* ist ein Fremdschlüssel.

Entity-Integrität

Primärschlüsselattribute sind immer bekannt, also nicht NULL.

Wird von allen RDBMS geprüft.

Referenzielle Integrität

Jeder Fremdschlüssel referenziert ein existierendes Tupel oder ist NULL.

Sollte von RDBMS geprüft werden, nicht von Anwendung. In der Praxis (leider) nicht immer verwendet.

Vom Relationalen Modell zum RDBMS

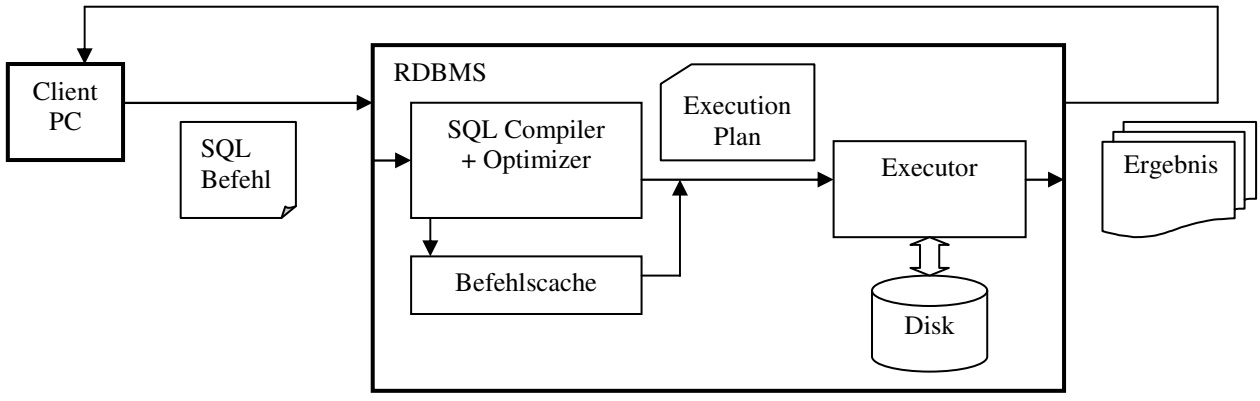
Die 12+1 RDBMS-Regeln von Codd

(siehe z.B. Buch von H. Sauer)

Damit sich ein konkretes RDBMS (= Relational Database Management System) 'Relational' nennen darf, muss es sämtliche der folgenden Regeln erfüllen.

0. **Grundregel**: jedes RDBMS muss die gesamte Datenbank entsprechend dem relationalen Datenmodell verwalten.
1. **Informationsregel**: Jede Information in einer relationalen Datenbank wird ausschließlich auf einer logischen Ebene und nur in genau einer Weise - durch Relationen - dargestellt.
2. **Garantierter Zugriff**: Jedes einzelne Datum (atomarer Wert) in einer relationalen Datenbank ist immer durch eine logische Kombination aus Relationsname, Primärschlüsselwert und Spaltenname erreichbar.
3. **Systematische Behandlung von fehlenden Informationen**: in einem RDBMS gibt es eine systematische Behandlung von unbekannten Werten (NULL), die unabhängig vom Attributtyp ist. Ein RDBMS stellt spezielle Operatoren zum Umgang mit unbekannten Werten (Test auf NULL) zur Verfügung, die ebenfalls unabhängig vom Attributtyp sind.
4. **Relationale Kataloginformation**: Die Beschreibung der Datenbank (Metadaten) erfolgt auf logischer Ebene genau wie die Darstellung gewöhnlicher Daten. Autorisierte Benutzer können Metadaten genau gleich wie normale Daten verwenden.
5. **Allumfassende Sprache**: (nicht von Codd) ein RDBMS muss zumindest eine Kommandosprache aufweisen, die sowohl Datenbankdefinition als auch Datenmanimulation und Datenabfrage erlaubt, die über eine wohldefinierte Syntax verfügt, die Transaktionen unterstützt, die Autorisierung unterstützt und die Integritätsregeln unterstützt (z.B. SQL).
6. **Datenänderungen durch Views**: ein RDBMS muss die Änderung von Daten über Views (=relationaler Ausdruck) unterstützen sofern so eine Änderung überhaupt möglich ist. Details später.
7. **High-Level Insert, Update, Delete**: Ein RDBMS muss mengenwertige Datenmanipulationsoperationen erlauben.
8. **Physische Datenunabhängigkeit**: Anwendungsprogramme bleiben logisch unbeeinträchtigt, wenn Veränderungen an der Speicherstruktur oder der Zugriffsmethode vorgenommen werden.
9. **Logische Datenunabhängigkeit**: Anwendungsprogramme bleiben logisch unbeeinträchtigt, wenn informationserhaltende Veränderungen an den Basisrelationen vorgenommen werden.
10. **Integritätsunabhängigkeit**: Integritätsbedingungen für eine Datenbank müssen mit Hilfe der Datenbeschreibungssprache definierbar sein und im Systemkatalog und nicht in den Anwendungsprogrammen abgelegt sein.
11. **Verteilungsunabhängigkeit:** Anwendungsprogramme bleiben logisch unbeeinträchtigt, wenn ein RDBMS verteilt oder wieder zusammengeführt wird.
12. **Unterwanderungsverbot:** in einem RDBMS darf es keine (imperative) low-level Sprache geben, mit der Integritätsbedingungen verletzt werden.

Bestandteile und Topologie eines RDBMS



SQL Einführung

SQL = Structured Query Languag (oft auch 'siequel' gesprochen).

Ab den 70-er Jahren bei IBM im Rahmen von Nachfolgern von 'System R' und SQUARE als Sprache SEQUEL entwickelt (Boyce und Chamberlain) und später umbenannt in SQL.

Von den meisten heutigen RDBMS unterstützt (DB2, Oracle, MS SQL, PostgresQL, mySQL, etc).

Umfasst DDL und DML (einschliesslich Query).

SQL ist leider nicht so weit standardisiert, dass man von SQL an sich sprechen kann. Vielmehr muss man immer dazusagen, welche Version eines bestimmten Standards oder eines bestimmten Herstellers man meint.

Weit verbreitet als Basis für herstellerspezifische Erweiterungen ist *SQL-92*, das auch *SQL-2* genannt wird und eine ANSI und ISO-Norm darstellt.

SQL-92 unterscheidet verschiedene *Conformance Levels*:
1 = Full, 2 = Intermediate, 3 = Entry Level (ähnlich SQL-1, ANSI-Standard von 1986).

Die meisten RDBMS sind Level 3, aber mit diversen Features aus anderen Levels und eigenen Erweiterungen ausgestattet.

Im Folgenden wird nach Möglichkeit SQL-92 verwendet. Auf Abweichungen und Fallen wird hingewiesen.

Lexikalische Struktur von SQL

Eine *Anweisung* (Statement) ist als lesbarer *Text* dargestellt, der aus einer Folge von *Wörtern* besteht.

Jedes Wort besteht aus einer Folge von *Zeichen*.

SQL ist *Case Insensitiv* (außer bei Strings).

Zwischen den Wörtern kann beliebiger *white space* (Blank, TAB, EOL) oder Kommentar /*...*/ stehen.

Wörter sind entweder:

- reservierte Schlüsselwörter mit Spezialbedeutung
- Namen von Tabellen, Attributen, etc.
- Spezialzeichen (Klammern, Komma, etc.)
- Operatoren (+, -, <, AND, OR, etc.)
- Literale (Konstante Werte, zB Strings 'abc', 'captain"s dinner' oder Zahlen 123, 123.456)

Syntax von SQL

Im Folgenden definiert unter Verwendung der EBNF-Grammatiknotation:

{X}	X kann beliebig of wiederholt werden, einschl. 0 mal
[X]	X ist optional, 0 oder 1 mal
(X)	Syntaktische Klammer
|	Alternativentrennung
=	Trennt Name von Rumpf
.	beendet Syntaxregel

Schlüsselwörter sind im Folgenden groß und so wie Operatoren und Spezialzeichen unter doppelte Hochkomma geschrieben.

Syntaxregeln werden im Folgenden mit einem führenden Paragraphenzeichen ('§') geschrieben.

DDL (Data Definition Language)

Umfasst Kommandos zur Schemadefinition.

Zentraler Ausgangspunkt ist die CREATE-Anweisung, die unter anderem zur Definition von Tabellen und deren Spalten verwendet wird.

§ CreateTableStat = "CREATE" "TABLE" Tabellenname "("
AttributDef {"," AttributDef}
["," TableConstraints]
")".

Der Tabellenname entspricht dem Namen einer Relation und ist im einfachsten Fall ein nicht weiter zerlegbarer Bezeichner (z.B. Artikel, Lieferant). Ein Name kann auch das Zeichen '_' enthalten.

In der Praxis werden aber oft innerhalb eines RDBMS mehrere Namensräume benötigt, die voneinander unabhängig die Verwendung von gleichen Tabellennamen erlauben. Beispiel: Test und Produktionssystem in einer RDBMS. Zur Weiterentwicklung einer Anwendung braucht man oft eine Testdatenbank, die sehr ähnlich wie die Produktionsdatenbank ist. Zur Vereinfachung des Installationsaufwandes soll *ein* Datenbankserver *mehrere* Datenbanken verwalten können.

Lösung: Unterteilung des Namensraumes der Tabellen in sogenannte 'Kataloge'.

Weiters wird ein Katalog z.B. das Testsystem oft von mehreren Programmierern verwendet, die unabhängig voneinander Tabellen definieren wollen. Es ist daher der Namensraum eines Katalogs weiter untergliedert in separate Namensräume pro Benutzer. Diese Namensräume werden in SQL (leider) auch 'Schema' genannt.

Der vollständige Name einer Tabelle ergibt sich somit in vielen RDBMS als Katalog.Schema.Tabelle, z.B. Testdb.Templ.Artikel. Die allgemeine Form ist in folgender Syntax-Regel zusammengefasst:

§ Tabellenname = [[Katalogname "."] Schemaname "."] Name.

Kataloge (=Datenbanken) können auch mit DDL erzeugt werden (z.B. CREATE DATABSE ...). Allerdings gibt es dafür keinen Standard und ein RDBMS stellt meist benutzerfreundliche Tools dafür zur Verfügung. Das gleiche gilt für Schematas (z.B. CREATE SCHEMA ...).

§ AttributDef = Name Typ ["DEFAULT" Const] {ColConstraint}.

§ ColConstraint = "NOT" "NULL" | "PRIMARY" "KEY"
| "REFERENCES" Tabellenname [OnDelete].

§ OnDelete = "ON" "DELETE" ("CASCADE" | "SET" "NULL").

Die Attributdefinition besteht aus Name, Typ und optional einem Default-Wert sowie Zusatzbedingungen, genannt *Constraint*.

Ein Beispiel dafür ist die NOT NULL Option. Wenn nicht angegeben ist ein Attribut *Nullable*, außer es ist Teil des Primärschlüssels. Primärschlüsselfelder dürfen niemals Nullable sein wegen Entity-Integrität.

Primär- und Fremdschlüsseldefinition auf Attributebene nur für einstellige Schlüssel. Sonst Tabellen-Constraint.

Mit der Option ON DEKETE kann festgelegt werden, was passiert, wenn das durch einen Fremdschlüssel referenzierte Tupel gelöscht wird. Mit CASCADE wird das referenzierende Tupel mitgelöscht, mit SET NULL wird die Referenz auf NULL gesetzt. Fehlt die Option, so erfolgt einen Fehlermeldung. Damit wird die referenzielle Integrität auch beim Löschen in jedem Fall gewahrt.

Die wichtigsten (von den meisten RDBMS unterstützten) Datentypen sind:

§ Typ = "CHAR" "(" Const ")" | "VARCHAR" "(" Const ")"
| "INTEGER" | "SMALLINT" | "BYTE" | "BIGINT" | "DECIMAL" "(" Const ["," Const] ")"
| "FLOAT" | "REAL" | "DOUBLE" "PRECISION" | "DATETIME" | "IMAGE".

CHAR(N)	Zeichenkette der Länge N, rechts aufgefüllt mit Leerezichen.
VARCHAR(N)	Zeichenkette mit maximaler Länge N, nicht aufgefüllt. Länge meist limitiert mit Seitengröße gemäß physischer Org. des RDBMS (z.B. 8 KB).
INTEGER	Ganze 32-bit Zahlen mit Vorzeichen, auch INT genannt.
SMALLINT	Ganze 16-bit Zahlen mit Vorzeichen.
BYTE	Ganze 8-bit Zahlen mit/ohne Vorzeichen je nach RDBMS, manchmal auch TINYINT genannt.
BIGINT	In letzter Zeit häufig auch 64-bit Ganzzahlen mit Vorzeichen.
DECIMAL(P, S)	Dezimalzahl aus insgesamt maximal P Dezimalziffern (Precision), davon S nach dem Dezimalpunkt (Scale). Auch NUMERIC genannt.
REAL, FLOAT, DOUBLE PRECISION	Gleitkommazahl, 32 bzw. 64 bit.
DATETIME	Datum und Uhrzeit. Mmanchmal auch TIMESTAMP genannt, Vorsicht vor Verwechslung mit MS SQL-server Datentyp TIMESTAMP, der etwas anderes bedeutet. DATETIME Literale meist geschrieben als Zeichenkette der Form 'YYYY-MM-DD HH:MM:SS'. Oft auch Millisekunden- oder Mikrosekundenangabe möglich nach einem '.'.
IMAGE	Beliebige Folge von Bytes, nicht interpretiert. Auch BLOB (Binary Large Object) oder RAW genannt. Kann nur eingeschränkt verwendet werden. Z.B. zum Speichern von Dateien in RDBMS.
TEXT	Beliebig langes Textfeld. Auch CLOB (Character Large Object) oder LONGVARCHAR genannt.

Häufig gibt es weitere Typen, z.B. MONEY, DATE, TIME, GUID, ROWID, TIMESTAMP, BIT, etc. Immer auch die Wertebereiche eines konkreten RDBMS überprüfen.

Für Primärschlüsselspezifikation gibt es in den RDBMS auch andere oder erweiterte Notationen. Das gleiche gilt für die Foreign Key Definition.

Primär- und Fremdschlüssel für mehrstellige Schlüssel werden (meist) als Tabellen-Constraints angegeben.

§ TableConstraints = ["PRIMARY" "KEY" "(" AttrList ")"]

{"FOREIGN" "KEY" "(" AttrList ")"

"REFERENCES" "(" Tabellenname ")" [OnDelete]}

§ AttrList = Attributname {"," Attributname}.

Beispiele:

```
CREATE TABLE Lieferant (
  lifnr INTEGER,
  name VARCHAR(40),
  plz SMALLINT,
  ort VARCHAR(40),
  str VARCHAR(40),
  tel VARCHAR(20),
  PRIMARY KEY (lifnr)
)
```

```
CREATE TABLE Artikel (
  artnr INTEGER,
  bezeichnung VARCHAR(40),
  preis DECIMAL(10, 2) NOT NULL,
  menge INTEGER DEFAULT 0,
  lifnr INTEGER,
  PRIMARY KEY (artnr)
  FOREIGN KEY (lifnr) REFERENCES (Lieferant)
)
```

```
Oder einfacher:
```

```
CREATE TABLE Artikel (
  artnr INTEGER PRIMARY KEY,
  bezeichnung VARCHAR(40),
  preis DECIMAL(10, 2) NOT NULL,
  menge INTEGER DEFAULT 0,
  lifnr INTEGER REFERENCES Lieferant
)
```

Ändern des Schemas

Die ALTER TABLE Anweisung erlaubt in vielen RDBMS das nachträgliche Ändern eines Schemas bei gleichzeitiger Migration vorhandener Daten.

Mögliche Operationen sind das Hinzufügen oder Entfernen von Spalten, das Umbenennen von Spalten, das Ändern des Typs (z.B. Vergrößern eines VARCHAR) oder das Definieren von Primär- und Fremdschlüsseln.

Die Syntax ist hier meist RDBMS abhängig.

Beispiel für Foreign Key Definition hinzufügen in DB2:

```
alter table Artikel
   add constraint F_REFERENCE_1 foreign key (artikelnr)
      references Lieferant(artikelnr)
      on delete restrict on update restrict
```

Entfernen von Tabellen

Mit der DROP TABLE-Anweisung können Tabellen entfernt werden. Inhalt (Tupel, Daten) und Schema der Tabelle und werden gelöscht.

§ DropStat = "DROP" "TABLE" Tabellenname.

Beispiel: `DROP TABLE Artikel`

Benutzerrechte

Ein RDBMS verwaltet meist mehrere Benutzer (User) und deren Rechte. Die Details sind RDBMS spezifisch. Ein Benutzer ist aber in der DDL immer durch einen Namen identifiziert.

Mit der GRANT-Anweisung können einem Benutzer bestimmte Rechte an bestimmten Tabellen eingeräumt werden.

§ GrantStat = "GRANT" ("ALL" ["PRIVILEGES"] | Privileg {"," Privileg})
"ON" Tabellenname
"TO" ("PUBLIC" | User {"," User}).
§ Privileg = "SELECT" | "INSERT" | "DELETE"
| "UPDATE" ["(" AttrList ")"].

Beispiele:

```
GRANT ALL ON Artikel TO PUBLIC
GRANT SELECT, UPDATE(menge) ON Artikel TO Templ, Ulamec
```

DML (Data Manipulation Language)

Mit der INSERT-Anweisung kann eine Zeile in eine Tabelle eingefügt werden.

§ InsertStat = "INSERT" ["INTO"] Tabellenname ["(" AttrList ")"]
"VALUES" "(" Const {"," Const} ")".

Beispiele:

```
INSERT INTO Artikel (artikelnr, bezeichnung, preis, menge, lifnr)
VALUES (1, 'Winterreifen", 100, 17, NULL)
INSERT Artikel VALUES (1, 'Winterreifen", 100, 17, NULL)
```

Die Form ohne Namensliste kann verwendet werden, wenn Reihenfolge der Werte genau der Attributreihenfolge im CREATE TABLE entspricht. Achtung: nicht stabil gegenüber allfälligen Änderungen der Attributreihenfolge oder dem Hinzufügen von nullable Attributen!

Die Werte sind hier konstant. Es können Zahlen, Zeichenketten, etc. verwendet werden.

Fehlende Attribute werde mit dem DEFAULT Wert des Attributs belegt, oder mit NULL wenn keiner definiert ist. Ist NULL nicht erlaubt, so folgt eine Fehlermeldung.

Löschen von Zeilen

Mit der DELETE-Anweisung können Zeilen einer Tabelle gelöscht werden.

§ DeleteStat = "DELETE" ["FROM"] Tabellenname ["WHERE" Cond].

Löscht alle Tupel, für die die Bedingung Cond erfüllt ist.

Beispiel:

```
DELETE FROM Artikel WHERE menge = 0
```

Ändern von Zeilen

Mit der UPDATE-Anweisung können existierende Zeilen geändert werden.

§ UpdateStat = "UPDATE" Tabellenname "SET"
Attributname "=" Expr {"," Attributname "=" Expr}
["WHERE" Cond].

Die angegebenen Attribute werden auf die in Expr definierten Werte gesetzt.

Gilt für alle Zeilen, außer wenn WHERE angegeben ist, dann werden nur jene Zeilen geändert, für die Cond erfüllt ist.

Beispiel: Generelle Preiserhöhung

```
UPDATE Artikel SET preis = preis * 1.1
```

Selektive Preiserhöhung

```
UPDATE Artikel SET preis = preis * 1.2 WHERE lifnr = 77
```

Relationale Operatoren in SQL

Die SELECT-Anweisung erlaubt die Formulierung einzelner und zusammengesetzter Operationen der Relationenalgebra und tritt in vielfältiger Form auf, die im Folgenden schrittweise entwickelt wird.

§ Restriktion = "SELECT" "*" "FROM" Tabellenname ["WHERE" Cond].

Liefert alle Tupel aus der angegebene Tabelle, für die Cond erfüllt ist.

§ Projektion = "SELECT" ["DISTINCT"] SelList "FROM" Tabellenname ["WHERE" Cond].

Projektion in SQL liefert alle Tupel, außer bei Angabe von DISTINCT. SelList ist eine Liste von Ausdrücken, die im einfachsten Fall ein Attribut bezeichnen. Damit können auch neue (berechnete) Spalten hinzugefügt werden.

§ Produkt = "SELECT" AttrList "FROM" Tabellenname {"," Tabellenname} ["WHERE" Cond].

Bildet das Produkt der angegebenen Tabellen und wendet anschließend eine optionale Restriktion darauf an.

§ Tabellenname = [[Katalogname "."] Schemaname "."] Name [["AS"] Name].

Umbenennung ist erlaubt für Tabellenname und in SelList wodurch sich für SelList folgende Regel ergibt (Expr ist ein Ausdruck, der weiter unten genauer behandelt wird).

§ SelList = Expr [["AS"] Name] {"," Expr [["AS"] Name]}.

Beispiel:

```
SELECT artnr * 100 AS A100, A.bezeichnung As Artikelbezeichnung
FROM Artikel AS A
```

Die bisherigen Möglichkeiten der Select-Anweisung zusammengefasst ergibt:

§ SelectTerm = "SELECT" ["DISTINCT"] SelList "FROM" SelBase ["WHERE" Cond].

§ SelBase = Tabellenname {"," Tabellenname}.

Vereinigung von Relationen ist möglich durch den UNION-Operator, wodurch sich eine weitere Verallgemeinerung ergibt:

§ SelectExpr = SelectTerm {"UNION" ["ALL"] SelectTerm}.

UNION verbindet zwei mit SELECT gebildete Relationen. Duplikate werden entfernt, außer bei Angabe von ALL. Da UNION ALL in Kombination mit UNION nicht assoziativ ist, gibt es noch die Möglichkiet, eine explizite Klammerung durchzuführen.

§ SelectExpr = (SelectTerm | "(" SelectExpr ")") {"UNION" ["ALL"] (SelectTerm | "(" SelectExpr ")")}.

Sortierung der Ausgabe von SELECT

Die Select-Anweisung erlaubt auch die Angabe eines Sortierkriteriums für die Ausgabe, aber nur auf der äussersten Ebene (z.B. nicht bei Union-Argumenten). Daraus folgt die (vorläufige) Form der Select-Anweisung.

§ SelectStat = SelectExpr ["ORDER" "BY" Expr ["DESC"] {"," Expr ["DESC"]}].

Ausdrücke

Wie bereits erwähnt erlaubt SQL die Verwendung von beliebig zusammengesetzten Ausdrücken für die Berechnung von Werten (Expr) in SelList und für Bedingungen (Cond) der WHERE-Klausel. Als Operanden können Literale und Attributnamen verwendet werden. Runde Klammern werden für Vorrangregeln verwendet. Folgende Operatoren stehen zur Verfügung:

Vergleichsoperatoren (<, >, =, <>, <=, >=)

boolesche Operatoren (OR, AND, NOT)

arithmetische Operatoren (+, -, *, /, %)

Stringoperatoren (+, like), diverse Funktionen (substring, Datumsfunktionen, ISNULL, COALESCE, etc.), den CASE-Operator für bedingte Ausdrücke

IN, BETWEEN, IS NULL, IS NOT NULL

Behandlung von NULL in Ausdrücken:

Bei logischen Ausdrücken gilt eine 3-wertige Logik: true, false, unknown. Ein SELECT selektiert nur jene Tupel für die Cond true liefert.

Beispiel: wenn ein Integer-Attribut x unbekannt also NULL ist, so ist weder x = 0 noch x <> 0 wahr. x IS NULL ist wahr, x IS NOT NULL ist nicht wahr.

Nur die Operationen IS NULL und IS NOT NULL können verwendet werden um auf NULL zu testen. Vergleich (=, <>, ...) mit der Konstanten NULL ist unzulässig, zB. lifnr = NULL; korrekt: lifnr IS NULL.

Bei logischen Verknüpfungen (AND, OR) mit einem NULL-Operanden kann das Ergebnis true, false oder unknown sein, je nachdem ob das Ergebnis durch den anderen Operanden eindeutig ist. Der Ausdruck ((a > b) AND c = 42) ergibt zum Beispiel false bei a=1, b=2 und c IS NULL.

Bei allen anderen Ausdrücken gilt, dass NULL als Operand immer NULL als Ergebnis ergibt.

NULL ist fundamental verschieden von 0 oder ''. NULL heißt *unbekannt*.

unbekannt + 1 ergibt wieder unbekannt.

Vorsicht: die Regeln für NULL sind nicht in jedem RDBMS konsequent implementiert und manchmal sind sie per Option einstellbar.

Beispiel: Oracle kann NULL nicht von leeren Strings unterscheiden (eigentlich kein RDBMS nach Codd).

LIKE erlaubt einen Patternvergleich der Art: s like 'pattern'

Das Zeichen '%' in pattern steht für 0 oder mehr beliebige Zeichen, '_' steht für genau 1 beliebiges Zeichen.

LIKE ist case sensitiv, allerdings nicht bei jedem RDBMS. Manchmal einstellbar per Optionen.