SQL

Prädikate

wert kann eine Konstante sein oder eine Spalte repräsentieren.
Mit *tabellename.spalte* wählt man Spalten von anderen Tabellen.

Ausdruck	$wert_1 < wert_2$	$\{<, \leq, =, \geq, >\}$
Between	*wert* `BETWEEN` i `AND` j	
	$wert \geq i$ `AND` $wert \leq j$	äquivalent
In	*wert* `IN` $(item_1, item_2, \ldots, item_n)$	
	$wert = item_1$ `OR` $wert = item_2$ `OR` ... `OR` $wert = item_n$	äquivalent
	wert `IN` (DQL)	Kombination mit DQL
Exists	`EXISTS` (DQL)	
	`COUNT`$(DQL) > 0$	äquivalent
Like	*wert* `LIKE('%x%')`	x Teilwort von *wert*
Negation	`NOT` *Prädikat*	negiert das ganze

DDL: Data Definition Language

CREATE TABLE *tabellename* (

$$\left.\begin{array}{llll} Spaltenname_1 & Datentyp & Inline\ Constraint, & \\ \vdots & \vdots & \vdots & \\ Spaltenname_n & Datentyp & Inline\ Constraint, & \end{array}\right\} \begin{array}{r} spalte_1 \\ \vdots \\ spalte_n \end{array}$$

Out-of-line Constraints
);

```
ALTER TABLE  tabellename (

ADD COLUMN spaltename
DROP COLUMN spaltename )
```

DML: Data Manipulation Language

INSERT INTO *tabellename* ($spalte_1$, $spalte_2, \ldots,$ $spalte_n$) **VALUES** ($wert_1$, $wert_2$, ..., $wert_n$); **UPDATE** *tabellename* **SET** *spalte* = *wert*;

DQL: Data Query Language

Clauses

Zum wählen von Spalten

Clause	Beispiel
SELECT	AVG(*spalte*)
	SUM(*spalte*)
	COUNT(*spalte*)
	MIN(*spalte*)
	MAX(*spalte*)
	(CASE WHEN *'spalte erfüllt Bedingung'* THEN konstante END)
	kann beliebig wiederholt werden
FROM	FROM *tabellename(n)*
WHERE	WHERE $spalte(n)_i$ erfüllt Prädikat(e)
ORDER BY	ORDER BY $spalte(n)_k$ ASC/DESC

Zum wählen von Gruppen innerhalb Spalten (in Kombination mit Aggregatfunktionen)

Clause	Beispiel
SELECT	*grouped-by-spalte*
	AVG(*spalte*)
	SUM(*spalte*)
	COUNT(*spalte*)
	MIN(*spalte*)
	MAX(*spalte*)
FROM	**wie oben**
GROUP BY	GROUP BY $spalte_j$
HAVING	HAVING $spalte_i$ erfüllt Prädikat

SELECT ($spalte_1$, $spalte_2, \ldots,$ $spalte_n$) **FROM** *tabellename* **WHERE** *Bedingung* ;

Datentypen

BOOLEAN	True oder False
INTEGER	$\in [-32676, 32676]$
NUMBER(i, j)	Nummer mit i Stellen wovon j Nachkommastellen
CHAR(n)	genau n Characters
VARCHAR(n)	bis zu n Characters
CLOB	für große Texte
BLOB	Binary Large OBject (z.B. Bilder)

Constraints sind optional, allerdings sollte jede Tabelle einen PRIMARY KEY haben

NOT NULL	Wert darf nie NULL sein	**nur inline**
UNIQUE	Einzigartiger Wert in der Spalte	
	spalte UNIQUE	**inline**
	UNIQUE(*spalte*$_1$, ..., *spalte*$_n$)	**out-of-line**
PRIMARY KEY	Einzigartige Identifikation pro Zeile	
	spalte PRIMARY KEY	**inline**
	PRIMARY KEY(*spalte*$_1$, ..., *spalte*$_n$)	**out-of-line**
FOREIGN KEY	Einzigartige Identifikation einer anderen Tabelle	
	spalte FOREIGN KEY	**inline**
	FOREIGN KEY(*spalte*) REFERENCES ...	**out-of-line**
CHECK	Voraussetzungen die ein Wert erfüllen muss	
	CHECK(*spalte prädikat*)	**inline**
	CHECK(*prädikat*)	**out-of-line**
REFERENCES	Referenz zu einer anderen Tabelle	
	spalte REFERENCES *tabelle*	
	spalte REFERENCES *tabelle* ON DELETE SET NULL	
	spalte REFERENCES *tabelle* ON DELETE CASCADE	

JDBC

Verbindung öffnen

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
con = DriverManager.getConnection(URL, USER, PWD); // throws SQLException
```

Verbindung schließen

```
con.close(); // throws SQLException
```

SQL ausführen

```
Statement stmt;
try {
stmt = con.createStatement();
stmt.executeUpdate(query);
} catch (SQLException e) {
con.rollback();
} finally {
stmt.close();
}
```

Prepared Statement ausführen (Beispiel)

```
PreparedStatement insV = con.prepareStatement("INSERT␣INTO␣Tabelle␣VALUES␣(?,␣?)");

insV.setString(1, "irgendetwas");
insV.setInt(2, 34);
// setString/setInt/setDate

try {
stmt = con.createStatement();
insV.executeUpdate();
} catch (SQLException e) {
con.rollback();
} finally {
stmt.close();
}
```

Daten auslesen

```java
ResultSet rs;
PreparedStatement query = con.prepareStatement("SELECT spalte1, spalte2 FROM Tabelle WHERE
    spalte3 = ?");
query.setString(1, name);
rs = query.executeQuery();
if (rs.next()) { // while wenn mehrere Ergebnisse
var1 = rs.getInt("spalte1");
var2 = rs.getInt("spalte2");
}
rs.close();
query.close();
```

Relationale Algebra

S

s
a
b
c

U

t
$NULL$
2
3

$S(s,t), U(t,u)$

Symbole und Beispiele

Selektion σ	$\sigma_{Spalte\ Bedingung}(Tabelle)$
Projektion π	$\pi_{Spalte}(Tabelle)$
Zuweisung $\rho \leftarrow$	$\rho_{\text{neuer Tabellename}}(Tabelle)$
Umbenennung	$\rho_{neuer\ Spaltename \leftarrow\ Spalte}(Tabelle)$
Vereinigung $\cup$	$Tabelle_1 \cup Tabelle_2$
Mengendifferenz $-$	
Durchschnitt $\cap$	$Tabelle_1 \cap Tabelle_2 = Tabelle_1 - (Tabelle_1 - Tabelle_2)$
Karthesische Produkt $\times$	$Tabelle_1 \times Tabelle_2$
Division $\div$	
(inner) Join $\bowtie$	$(b,2,2,y)$
Left Outer Join $⟕$	$(a,NULL,\text{-},\text{-})(b,2,2,y)(c,2,\text{-},\text{-})$
Right Outer Join $⟖$	$(\text{-},\text{-},1,x)(b,2,2,y)(\text{-},\text{-},NULL,z)$
Full Outer Join $⟗$	$(a,NULL,\text{-},\text{-})(b,2,2,y)(c,3,\text{-},\text{-})(\text{-},\text{-},1,x)(\text{-},\text{-},NULL,z)$
Left Semi Join $\ltimes$	$(b,2)$
Right Semi Join $\rtimes$	$(2,y)$
Group by/Aggregate γ	$\gamma_{Spalte;\mathbf{count}(*)}(Tabelle)$
	$\gamma_{Spalte;\mathbf{sum}(*)}(Tabelle)$

Relationale Entwurfstheorie

Schema	$\mathcal{R}$
Ausprägung	R
Spalte(n)	$\alpha \subseteq \mathcal{R}$
Instanz	$r \in R$
Wert(e)	$r.\alpha$
Funktional abhängig (FD)	$\forall r,s \in R : r.\alpha = s.\alpha \Rightarrow r.\beta = s.\beta$

β ist funkt. abh. von α
Notation : $\alpha \rightarrow \beta$

Mehrw. Abh. (MVD) $\exists t1, t2 : t1.\alpha = t2.\alpha \Rightarrow \exists t3, t4 :$ Notation: $\alpha \twoheadrightarrow \beta$

- $t3.\alpha = t4.\alpha = t1.\alpha = t2.\alpha$
- $t3.\beta = t1.\beta, t4.\beta = t2.\beta$
- $t3.\gamma = t2.\gamma, t4.\gamma = t1.\gamma$

Menge aller Funkt. Abh. F

Superschlüssel α heißt Superschlüssel wenn $\alpha \rightarrow \mathcal{R}$

Kandidatschlüssel α heißt Kandidatschlüssel wenn α ein minimaler Superschlüssel ist

Voll Funkt. abh. $\alpha \rightarrow \beta \wedge \forall A \in \alpha : \neg\Big((\alpha \setminus \{A\}) \rightarrow \beta\Big)$ β ist voll funkt. abh. von α

Herleitungen

$$\frac{\beta \subseteq \alpha}{\alpha \rightarrow \beta}$$ Reflexivität

$$\frac{\alpha \rightarrow \beta}{\alpha \cup \gamma \rightarrow \beta \cup \gamma}$$ Verstärkung

$$\frac{\alpha \rightarrow \beta, \beta \rightarrow \gamma}{\alpha \rightarrow \gamma}$$ Transitivität

$$\frac{\alpha \rightarrow \beta, \alpha \rightarrow \gamma}{\alpha \rightarrow \beta\gamma}$$ Vereinigungsregel

$$\frac{\alpha \rightarrow \beta\gamma}{\alpha \rightarrow \beta, \alpha \rightarrow \gamma}$$ Dekompositionsregel

$$\frac{\alpha \rightarrow \beta, \gamma\beta \rightarrow \delta}{\alpha\gamma \rightarrow \delta}$$ Pseudotransitivitätsregel

Zerlegung einer Relation $\mathcal{R} \rightarrow \mathcal{R}_1, \ldots, \mathcal{R}_n$

Verlustlosigkeit $\mathcal{R} = \mathcal{R}_1 \bowtie \ldots \bowtie \mathcal{R}_n$

Abhängigkeitserhaltung $FD(\mathcal{R})+ = \Big(FD(\mathcal{R}_1) \cup \ldots \cup FD(\mathcal{R}_n)\Big)$

Normalformen

1. NF Jedes Attribut muss ein atomares Wertebereich haben

2. NF Eine Tabelle darf keine 2 Kandidatschlüssel haben

1. NF gilt

3. NF $\forall \alpha \rightarrow B, B \in \mathcal{R}$ und $\big(B \in \alpha \vee B$ ist prim $\vee\ \alpha$ ist Superschlüssel von $\mathcal{R}\big)$

2. NF gilt

Boyce Codd (3.5) NF $\forall \alpha \rightarrow \beta : \beta \subseteq \alpha \vee \alpha$ ist Superschlüssel von $\mathcal{R}$ BCNF $\Rightarrow$ 3. NF

4. NF $\forall \alpha \twoheadrightarrow \beta : (\beta \subseteq \alpha \vee \beta = R - \alpha) \vee \alpha$ ist Superschlüssel von $\mathcal{R}$

Transaktionen

T $b < r[x] < w[x] < c/a$

BoT $<$ *read* $x <$ *write* $x <$ Commit/Abort

Log Struktur [LSN[1], TransaktionsID[2], PageID[3], Redo[4], Undo[5], PrevLSN[6]]

[1]**LSN** Log Sequence Number, eine eindeutige Kennung des Log-Eintrags

[2]**TransaktionsID** Transaktionskennung der Transaktion die die Änderung durchgeführt hat

[3]**PageID** Kennung der Seite, auf der die Änderungsoperationen vollzogen wurde

[4]**Redo** Beschreibt wie die Änderung nachvollzogen werden kann

[5]**Undo** Beschreibt wie die Änderung rückgängig gemacht werden kann

[6]**PrevLSN** LSN des letzten Logeintrags

Compensation **L**og **R**ecord

CLR Struktur < LSN, TransaktionsID, PageID, Redo, PrevLSN, UndoNextLSN[7] >

[7]**UndoNextLSN** Der Logeintrag, der zurückgedreht werden muss

	Beispiel Log	**Transaktion**
Anfang	`[#1, T1, BoT, 0]`	`BOT`
	⋮	⋮
Lesen von var A in a_1		`r(A,a1)`
	⋮	⋮
Ändern von a_1		`a1:=a1+1`
	⋮	⋮
Schreiben von a_1 in A	`[#8, T1, PA, A+=1, A-=1, #1]`	`w(A,a1)`
	⋮	⋮
Systemabstürz und T_1 ist ein Loser	`<#8', T1, PA, A-=1, #8, #1>`	
	`<#1', T1, -, -, #8', 0>`	
Commit	`[#11, T1, commit, #8]`	`commit`
	⋮	⋮
Abort	wie bei commit	

Serialisierbarkeit

T_i Transaktion i

$r_i(A)$ Lesen des Objekts A in Transaktion i

$w_i(A)$ Schreiben des Datenobjekts A in Transaktion i

a_i Abort der Transaktion i

c_i Commit der Transaktion i

WaR $w_i(A)$ bevor $r_j(A)$ Konflikt $i \neq j$

RaW $r_i(A)$ bevor $w_j(A)$ Konflikt $i \neq j$

WaW $w_i(A)$ bevor $w_j(A)$ Konflikt $i \neq j$

RaR $r_i(A)$ bevor $r_j(A)$ (führt nicht zu Konflikte) $i \neq j$

$(T_i) \rightarrow (T_j)$ Es gibt ein Konflikt von i nach j

***lock*$_i$(A)** A ist nur für Transaktion$_i$ verfügbar

***unlock*$_i$(A)** A ist freigegeben

2PL Jedes Objekt das benutzt werden soll muss vorher gesperrt werden
Fordert keine Sperre die sie schon besitzt
Bei EoT muss eine Transaktion alle Sperren zurückgeben
Die Transaktion hat eine Wachstums bzw. Schrumpfphase bzgl. #Sperren

Strenges 2PL Alle Sperren werden bis zu EoT gehalten

SI Möglich wenn $\bigcap_i WriteSet(T_i) = \emptyset$

LockX(A) Reserviert A, oder falls A schon reserviert, wartet bis A freigegeben wird

Unlock(A) Gibt A frei

Deadlock Eine Situation wo Transaktionen auf einanders Freigabe warten
Zu Erkennen wenn es eine Zyklus gibt in der Graph

IO

Access time $t :=^1 t_s +^2 t_r +^2 t_{tr}$

[1]**Seek time** Bewege den Arm zur gewünschten Spur

[2]**Rotational delay** Warte darauf bis der Block/Sektor zum Lesekopf rotiert ist

[3]**Transfer time** Lese/Schreibe Daten

Paritätsfunktion $f(x) = \begin{cases} 0 & x \in \{0,1\}^n \text{ ist gerade} \\ 1 & x \in \{0,1\}^n \text{ ist ungerade} \end{cases}$

RAID 0 Anzahl von Platten die als eine Große Platte betrachtet wird
Hohes Ausfallrisiko

RAID 1 Speichert Daten auf mindestens 2 verschiedene Platten
Doppelte Lesegeschwindigkeit

RAID 2 Bit-level striping with dedicated Parity

RAID 3 Eine zusätsliche Platte wird verwendet für berechnete Parität für der Hamming Error Correct Code

RAID 4 Wie RAID3, nur wird die Parität für Blöcke (statt Bytes) berechnet

RAID 5 Wie RAID4, nur werden die Paritätsblöcke verteilt über mehrere Platten
IO Parallel nur zum Lesen, überlebt N-1 Plattenverluste, Spiegelung/Replikation

B$^+$-Bäume

B$^+$-Bäume sind immer balanziert.

Algorithmen

Data: F, α
Result: *result*
$result \leftarrow \alpha$,
while *result changed* **do**
  foreach $\beta \to \gamma \in F$ **do**
    if $\beta \subseteq result$ **then**
      $result \leftarrow result \cup \gamma$
  end
end

Algorithm 1: AttrHülle

Data: F
Result: F
`// linksreduktion`
foreach $\alpha \to \beta \in F$ **do**
  foreach $A \in \alpha$ **do**
    if $\beta \subseteq attrH\ddot{u}lle(F, \alpha - A)$ **then**
      $F \leftarrow F - \alpha \to \beta$
      $F \leftarrow F \cup \alpha - A \to \beta$
  end
end
`// rechtsreduktion`
foreach $\alpha \to \beta \in F$ **do**
  foreach $B \in \beta$ **do**
    if $B \in AttrH\ddot{u}lle(F - (\alpha \to \beta) \cup \alpha \to (\beta - B), \alpha)$ **then**
      $F \leftarrow F - \alpha \to \beta$
      $F \leftarrow F \cup \alpha \to \beta - B$
  end
end
`// vereinigungsregel anwenden auf` F`;` $(\frac{\alpha\to\beta, \alpha\to\gamma}{\alpha\to\beta\gamma})$

Algorithm 2: Kanonische Überdeckung (KanÜb)

Data: $\mathcal{R}, F$
Result: $\mathcal{R}_1, \ldots, \mathcal{R}_n$
$Fc \leftarrow Kan\ddot{U}b(F)$ `// Schritt 1`
`// Schritt 2`
foreach $\alpha \to \beta \in Fc$ **do**
  $\mathcal{R}\alpha \leftarrow \alpha \cup \beta$
  foreach $\alpha' \to \beta' \in Fc$ **do**
    `// ordne F`α `alle FD's von` $\mathcal{R}\alpha$ `zu`
    if $\alpha' \cup \beta' \subseteq \mathcal{R}\alpha$ **then**
      $F\alpha \leftarrow F\alpha \cup \alpha' \to \beta'$
  end
end
`// Schritt 3`
if $\nexists \alpha \in \mathcal{R}_i : \alpha$ *Kandidatenschlüssel von* $\mathcal{R}$ **then**
  $\mathcal{R}_k \leftarrow k$ `//` $k \in \mathcal{R} : k$ `ist Kandidatenschlüssel`
  $F_k \leftarrow \emptyset$
`// Schritt 4`
foreach $\mathcal{R}\alpha : \exists \mathcal{R}\beta : \mathcal{R}\alpha \subseteq \mathcal{R}\beta$ **do**
  $\mathcal{R}\alpha \leftarrow \emptyset$
end

Algorithm 3: Synthesealgorithmus

Data: $\mathcal{R}, F$
$Fc = Kan\ddot{U}b(F)$
foreach $\alpha \to \beta \in F$ **do**
  if $\neg(\alpha$ *ist Superschlüssel* $\vee\ \beta \subseteq \alpha)$ **then**
    return *false*
end
return *true*

Algorithm 4: Zur Bestimmung ob $\mathcal{R}$ in BCNF ist

```
Data: k, node
if node is a leaf then
    return node;
switch k do
    case k < k_0 do
        return tree_search(k, p_0)
    end
    case k_i ≤ k < k_{i+1} do
        return tree_search(k, p_i)
    end
    case k_{2d} ≤ k do
        return tree_search(k, p_{2d})
    end
end
```

Algorithm 5: tree_search

```
Data: k, rid, node
if node is a leaf then
    return leaf_insert(k, rid, node);
switch k do
    case k < k_0 do
        ⟨sep, ptr⟩ ← tree_insert(k, rid, p_0)
    end
    case k_i ≤ k < k_{i+1} do
        ⟨sep, ptr⟩ ← tree_insert(k, rid, p_i)
    end
    case k_{2d} ≤ k do
        ⟨sep, ptr⟩ ← tree_insert(k, rid, p_{2d})
    end
end
if sep is null then
    return ⟨null, null⟩
else
    return split(sep, ptr, node)
```

Algorithm 6: tree_insert

```
Data: k, rid, node
if node is a leaf then
    return leaf_insert(k, rid, node);
switch k do
    case k < k_0 do
        ⟨sep, ptr⟩ ← tree_insert(k, rid, p_0)
    end
    case k_i ≤ k < k_{i+1} do
        ⟨sep, ptr⟩ ← tree_insert(k, rid, p_i)
    end
    case k_{2d} ≤ k do
        ⟨sep, ptr⟩ ← tree_insert(k, rid, p_{2d})
    end
end
if sep is null then
    return ⟨null, null⟩
else
    return split(sep, ptr, node)
```

Algorithm 7: tree_insert