Begriffe

Datenbanksystem (DBS) Datenbankmanagementsystem DBMS + $n\cdot$ Datenbasen/Datenbestände/Datensätze (=strukturierte Sammlung von Daten)

Anforderungen an DBMS Redundanzfreiheit + Datenintegrität

Datenintegrität Datenkonsistenz(logische Widerspruchsfreiheit der Daten), Datensicherheit(Schutz vor physischem Verlust) & Datenschutz(Zugriffe)

Datenmodelle Hierarchisch, Netzwerk, Relationen, Postrelational: Objektrelational (Methoden, Tabellenvererbung, ~~1. NF~~), Objektorientiert (analog Programmiersprache)

DBMS-Funktionen Transaktionen, Mehrbenutzerbetrieb, Sicherheit, Backup & Recovery, Datenkatalog, SQL

Architektur

Tier *1-Tier*: DBMS im selben Prozess wie Client (MS Access), *2-Tier*: Client, Server mind. prozessmässig getrennt

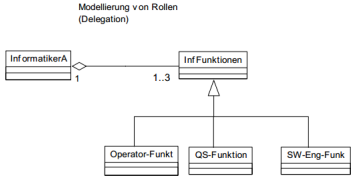
3-Ebenen Modell *externe Ebene*: Sicht auf Teilmenge der DB, Applikationen, externes Schema *konzeptionelle/logische Ebene*: konzeptionelles Schema/Modell/Entwurf(UML), logisches Modell/Datenbankschema *interne Ebene*: physische Datenstruktur, internes Schema (DDL)

UML

Aggregation: ◊ Komposition: ♦
Abstrakte Klassen
Objektname : Klassenname
$[visibility]AttrName[multiplicity][: type][= initial - value]$
$Name[0..1] : String, Salaer : float = 1000.0$

Generalisierung/Spezialisierung

disjunkt: 1 Objekt→Instanz einer Unterklasse {disjoint}
überlappend: 1 Objekt→mehreren Unterklassen {overlapping}
vollständig: Subklassen spezifiziert, keine zusätzlichen {complete}
unvollständig: nicht alle, zusätzliche erlaubt {incomplete}



Entitätstyp: Klasse **Entitätsmenge:** Alle Objekte einer Klasse, Relation **Krähenfüsschen:** c=optional, n/m=mehrere

Relationales Modell

Entitätsintegrität Kein Primärschlüsselattribut `NULL`

Referentielle Integrität Fremdschlüsselattribute ≠ `NULL`, wenn Primärschlüssel existiert

Student (id INTEGER, lang TEXT(3) NOT NULL UNIQUE, *abt* INTEGER NOT NULL REFERENCES TableB(tableB_id))

Redundanzen Mutationsanomalien: Einfüge-, Änderungs-Löschanomalie

Normalformen 1. NF: Wertebereiche der Attribute atomar, 2. NF: Jedes Attr. voll funktional abhängig von jedem Schlüsselattr., 3. NF: Kein Nichtschlüsselattr. von irgendeinem Schlüssel transitiv abhängig, BCNF: Schlüsselkand. {Name, Sportart}{Name, Verein}, Verein → Sportart nicht erlaubt!!!

Regeln optionale Assoziationen (0..*)→wenige?→Zwischentabelle, Assoziative Klassen: Zwischentabelle + die beiden fk = pk

SQL Data Definition Language

```
CREATE DATABASE foo WITH OWNER = 'bar';
CREATE INDEX indexname ON tbl_name (attr.);

CREATE TABLE Angestellter (
PersNr    INTEGER NOT NULL, --Column Constraint
Name      VARCHAR(20) NOT NULL,
Tel       TEXT NULL,
Salaer    FLOAT NOT NULL,
EintrittsDatum DATE DEFAULT CURRENT_DATE,
--Table Constraints
PRIMARY KEY (PersNr, Name),
CHECK (Salaer BETWEEN 1000 AND 20000)
);

ALTER TABLE team
 ADD CONSTRAINT constraintbezeichnung
       FOREIGN KEY (fk_stadion) REFERENCES
       stadion (stadion_id); --Foreign Keys immer so
```

Referentielle Integrität B abhängig von A
`ON [DELETE|UPDATE] CASCADE`
Tupel in B wird ebenfalls gelöscht/Referenz wird geändert
`ON [DELETE|UPDATE] RESTRICT`
Solange Referenz besteht, kann A nicht gelöscht werden (default)
Änderung wird nicht ausgeführt (default)
`ON [DELETE|UPDATE] SET NULL`
Fremdschlüsselattr. in Tupel B wird auf NULL gesetzt
`ON [DELETE|UPDATE] SET DEFAULT`
Fremdschlüsselattr. in Tupel B wird auf den Defaultwert gesetzt

SQL Data Manipulation Language

```
INSERT INTO Abteilung (Name, AbtNr) VALUES ('F&E', 20);
INSERT INTO tbl_name (attr.) SELECT ...;
UPDATE Abteilung SET abtnr=21 WHERE abtnr=20;
```

Queries

`SELECT DISTINCT...` Duplikate werden unterdrückt

Projektion $\pi_{\text{foo,bar}}(\text{tbl\_name})$
`SELECT foo, bar FROM A;`
Identische Tupel werden eliminiert

Selektion $\sigma_{\text{ID}<10}(tbl\_name)$
`SELECT * FROM A WHERE ID < 10;`

Kartesisches Produkt, CROSS JOIN $R_1 \times R_2$
`SELECT * FROM R1, R2; SELECT * FROM R1 CROSS JOIN R2;`

Vereinigung $A \cup B$
`SELECT * FROM A UNION [ALL] SELECT * FROM b;`
`UNION ALL`→Duplikate werden nicht entfernt
Schemata müssen gleich sein!

Differenz, Komplement $A - B$
`SELECT * FROM A [EXCEPT] SELECT * FROM B`
Schemata müssen gleich sein!

Umbenennung $\rho_{\text{bar}\leftarrow\text{foo}}(\text{tbl\_name})$
`SELECT foo AS bar FROM ...;`

Durchschnitt $A \cap B$
`SELECT * FROM A INTERSECT SELECT * FROM B;`
Schemata müssen gleich sein!

Unterabfragen

```
...WHERE Wohnort [NOT] IN ('Luzern', 'Zug', 'Horw')...
...WHERE [NOT] EXISTS(SELECT * ...)
WHERE attr. [<|>|...] [ANY|ALL] (SELECT * ...)
```
Vergleiche mit `NULL` (x > `NULL`) immer UNKNOWN
`SELECT * FROM (SELECT * FROM ...) AS ["]alias["];`

Aggregationsfunktionen

`MAX, MIN, AVG, SUM, COUNT`
`NULL`-Werte werden ignoriert (Ausnahme: `COUNT`)
`SELECT [Aggrfkt.](attr.) FROM tabl GROUP BY group, subgroup`
`GROUP BY...HAVING [Aggrfkt.](...)...;`
→analog `WHERE`, aber für Aggregationsfunktionen

Window functions

`SELECT [Aggrfkt.](attr) OVER (PARTITION BY ... ORDER BY...)`
Mit `PARTITION BY` wird gruppiert
`lag|lead (attr., offset, default) OVER (ORDER BY...)`
vorderer/hinterer Wert innerhalb der Partition wird ausgegeben
Defaultmässig offset=1, default=`NULL`

Common Table Expression (CTE)

`WITH tmp AS (SELECT...) SELECT * FROM tmp;`
Rekursiv:

```
WITH RECURSIVE cte (spalten) AS
   (Ursprungsselect UNION ALL Rekursionsselect)
SELECT spalten FROM cte WHERE ...;
```

Beispiel:

```
WITH RECURSIVE unter ( persnr, name, chef ) AS
   (SELECT A.persnr, A.name, A.chef FROM angestellter A
    WHERE A.chef = 1010    UNION ALL
    SELECT A.persnr, A.name, A.chef FROM angestellter A
    INNER JOIN unter B on B.persnr = A.chef
) SELECT * FROM unter ORDER BY chef, persnr;
```

Views

```
CREATE VIEW AngPublic (Persnr, Name, Tel, Wohnort) AS
SELECT Persnr, Name, Tel, Wohnort
FROM Angestellter;
```

Bedingungen für updatable: nur 1 `FROM`-Eintrag, kein `WITH,DISTINCT,GROUP BY,HAVING, LIMIT,OFFSET,UNION,INTERSECT, EXCEPT`, keine Aggregationen, window function

Joins

Theta-Join $\sigma_{=,>,<,<=,>=,<>}(R_1 \times R_2) = \bowtie_{=,>,...}$
Equi-Join $\sigma_{a.x=b.y}(A \times B)$
`NATURAL JOIN` Automatisch alle gleichnamigen Spalten verglichen und gejoint
Linker Semi-Join $\ltimes \pi_{\text{SchemalinkerRelation}}$(`NATURAL JOIN`)
Inner Join

`SELECT * FROM a [INNER] JOIN b ON a.x = b.y;`
`INNER` muss nicht zwingend geschrieben werden
Left (Outer) Join ⟕

`SELECT * FROM a LEFT [OUTER] JOIN b ON a.x = b.x;`
`OUTER` muss nicht zwingend geschrieben werden
right → dito
Full (Outer) Join ⟗

`SELECT * FROM a [FULL] OUTER JOIN b ON a.x = b.x;`
`FULL` muss nicht zwingend geschrieben werden

`...JOIN LATERAL..`damit kann tabellenübergreifend auf Attr. zugegriffen werden

Löschen

`DROP TABLE Angestellter CASCADE;`
Löscht zusätzlich alle Views + FK-Constraints
`TRUNC[ATE] TABLE tbl_name`
Löscht nur der Inhalt der Tabelle
`DELETE FROM Abteilung WHERE abtnr=21;`
Einzelne Tupel löschen

Varia

`COALESCE(NULL, NULL, 'Hallo')`
erster Nullwert zurück→Hallo wird zurückgegeben
`ROUND(Zahl, Anz. Stellen)`
Ohne Angabe der Anz. Stellen→keine Nachkommast.
`CREATE TEMPORARY TABLE mytable (...)`→besteht bis Ende der Transaktion/Session
`ORDER BY [attrn., Spaltennr.] [ASC|DESC];`
Default: ASC (aufsteigend, kleinster Wert zuerst)
`LIKE 'Zu\%'` % = 0 bis * Zeichen
`LIKE '____'` _ entspricht genau 1 bel. Zeichen

SQL Data Control Language

```
CREATE ROLE user/group WITH  LOGIN PASSWORD 'password'
[CREATEDB NOCREATEROLE];

ALTER USER name WITH [CREATEDB ...]
DROP ROLE name;

GRANT [SELECT|INSERT|UPDATE|...] ON tbl_name
TO role [WITH GRANT OPTION]
```

`[WITH GRANT OPTION]`→Rechte können weitergegeben werden
`REVOKE [Recht] ON tbl_name FROM rolename`

Transaktionen

Atomicity (Vollständig/gar nicht), **c**onsistency (konsistentem Zustand→n. konst. Zustand, **I**solation(Trans. von anderen isoliert), **D**urability (persistente Änderungen)

SQL Transaction Control Language

`BEGIN TRANSACTION;` `SET TRANSACTION ISOLATION LEVEL [...]` `COMMIT TRANSACTION;` `ROLLBACK TRANSACTION;` `SAVEPOINT name;` `RELEASE SAVEPOINT name;` `ROLLBACK TO SAVEPOINT name;`

Konfliktpaare

Zwei Transaktionen A,B — zwei Variablen
$(r_A(x), r_B(y))$→kein Konflikt
Alle anderen, wenn x == y→Konflikt!

Pessimistisches Verfahren/Sperrprotokolle

Exclusive Lock(x): RW, Shared Lock(s): R (mehrere gleichzeitig möglich)
2 Phase Locking: Growing: Sperren, Shrinking Phase: Sobald Transaktion ein Lock freigegeben hat→keine weiteren Bezüge, Am Schluss Freigabe aller Locks, -Cascading Rollback -Deadlocks
Strict 2PL: Alle gehaltenen Sperren nach Ende der Transaktion freigegeben +kein Cascading Rollback

Isolationlevels

(-): unmöglich mit cursor stability

Level	Dirty	Fuzzy/Nonrep.	Lost upd.	Phantom
uncommitted	-	-	(-)	-
committed	✓	-	(-)	-
Repeatable	✓	✓	✓	-
Serializable	✓	✓	✓	✓

Dirty Read: liest uncommited data von anderer Transaktion **Fuzzy/Nonrepeatable Read**: liest bereits gelesene Daten nochmals gleich ein, Daten wurden jedoch geändert **Lost Update**: Zwei Transaktionen verändern Info, erste durch zweite überschrieben **Phantom Read**: Liest gleiche Daten mehrmals, neue oder gelöschte Werte
committed: Sicht eines Snapshots vor Query (nur committed data ersichtlich), Änderungen innerhalb Transaktion ersichtlich **Repeatable**: Sicht eines Snapshots vor Transaktion, Änderungen innerhalb Transaktion ersichtlich

Snapshot Isolation (SI)

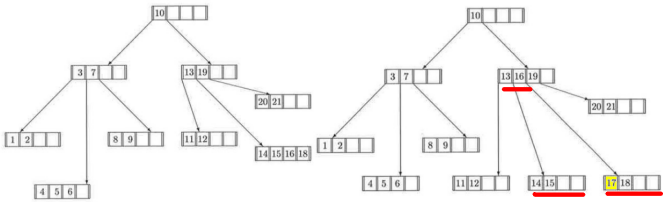
Änderungen→Objekte = Snap?Nein→Rollback
Keine Cascading Rollbacks, Deadlocks möglich

Multi-Version Concurrency Control (MVCC)

Schreiben: erzeugt neue Version, lesen: immer von aktuellster, Schreiber blockieren andere Schreiber

Index

Grad = k, k = $\frac{m}{2}$ = Min. Anzahl an Einträgen,
m = 2k = Max. Anzahl an Einträgen, max 2k+1 Unterknoten
17 in B-Baum einfügen:



Indexalg: ISAM, B-Bäume, B+-Bäume, Hash, BRIN, Bitmap(Bitmuster) **Indexvarianten**(~~*in Postgres*~~): Clustered Index, Primär Index, *Sekundär-Index*, zusammengesetzter Index, *partieller Index*, funktionaler Index, mehrstufige, mehrdimensionale

JDBC

Treibertypen

1. JDBC-ODBC-Brücke, 2. Native plattformeigene JDBC-Treiber, 3. Universeller JDBC-Treiber (Java), 4. Direkte Netzwerktreiber (Java)

```
try (Connection connection = DriverManager.getConnection
 ("jdbc:postgresql:mydb", username, password)) {

try (Statement statement = connection.createStatement()) {
 ResultSet resultSet = statement.executeQuery
 ("SELECT * FROM Account");
 while (resultSet.next()) {/*Start: vor erster Zeile*/
 String owner = resultSet.getString("Owner");/*oder über Index*/
 int balance = resultSet.getInt("Balance");/*NULL wird zu 0*/
 }}
} catch (SQLException e) {
}/*Keine Verbindung, Login fehlgeschlagen*/

PreparedStatement statement = connection.prepareStatement(
"UPDATE Account SET Balance = Balance + ? WHERE Owner = ?");
statement.setInt(1, -100);
statement.setString(2, "Bob");
statement.execute();
```

SQL-Injection-`'; UPDATE coffees SET price=price/2; --`

Transaktionen

```
connection.setAutoCommit(false);
connection.setTransactionIsolation(TRANSACTION_SERIALIZABLE)
/*Impliziter Transaktionsbeginn (erstes Statement)*/
connection.commit();
connection.rollback();/*Für Exception-Handling*/
```

MetaData

`connection.getMetaData()`: SQL-Sprachumf., DB-Produktname, Driver-Eig., Datentypen, Eigenschaften Transaktionen, ...
`resultSet.getMetaData()`→ResultSetMetaData

Scrollable/Updateable ResultSet

`createStatement()`-Parameter: `TYPE_FORWARD_ONLY`: kann nicht scrollen `TYPE_SCROLL_INSENSITIVE`,`TYPE_SCROLL_SENSITIVE`: reflektiert [keine] Änderung von anderen, `CONCUR_READ_ONLY`, `CONCUR_UPDATABLE`

```
rs.moveToInsertRow(); rs.updateString("Owner"); rs.insertRow();
```

Batch Updates

Default: jedes Statement = 1 Roundtrip, Batch Updates sammelt

```
statement.addBatch("INSERT|UPDATE|DELETE");
statement.addBatch("CREATE|DROP|ALTER");
int [] rowUpdateCount = statement.executeBatch();
```