Datenbanken Cheat Sheet

Begriffe

Information (lat. „informatio“, Form oder Gestalt geben) ist nicht einheitlich definiert. Im allgemeinen wird darunter „übertragenes Wissen“ verstanden.

Daten (lat. „datum“, gegebenes) ebenfalls nicht einheitlich definiert. Grundsätzlich codierbare Angaben über Dinge oder Sachverhalte, die gespeichert und verarbeitet werden können. Die jährlich produzierte Datenmenge liegt bei ca. 10 Zettabytes (10^{21} Bytes)

strukturierte Daten haben eine explizite Struktur und sind in der Minderheit (Bsp. Tabellen)

unstruktierte Daten haben keine explizite Struktur (ggf. implizite Struktur aufgrund einer zugrunde liegenden Grammatik. (Bsp. Text, Bilder, Filme, ...)

semistrukturierte Daten sind nur zum Teil strukturiert (Bsp. XML)

Dateisystem vs. DBMS

Dateisystem

Daten werden in Dateien vom Betriebssystem verwaltet. Anwendungen lesen/schreiben die Daten direkt. (Office-Anwendungen, Buchhaltungssoftware, ...)

- \+ einfach, angepasst, effizient
- \+ keine Rücksicht auf andere
- \+ proprietäre Formate möglich
- – Verwendung für unterschiedliche Zwecke problematisch
- – i.d.R. Strukturänderung ⇒ Programmänderung
- – Synchroner Zugriff aufwändig zu realisieren
- – Abgestufte Zugriffsrechte aufwänding zu realisieren
- – Daten häufig mehrfach gespeichert
- – Datenaustausch, -integration komplex

Datenbanksysteme

Verwalten und nutzen (sehr) grosse Datenmengen. Der Zugriff ist deklarativ und mengenorientiert. Die Daten werden einmalig und zentral definiert. Wichtige Aufgaben (Integritätskontrolle, Redundanzverwaltung, Zugriffskontrolle und -optimierung, synchroner Zugriff, zentrale Datensicherung und Wiederherstellung) können so automatisiert werden. Allerdings sind Aufbau und Betrieb eines Datenbanksystems anspruchsvoll und teuer.

Datenunabhängigkeit (DU)

Logische DU Anwendungsprogramme müssen die logische Gesamtstruktur nicht kennen, um spezifische Verarbeitungen vorzunehmen ⇒ sie sind von Datenbankschema-Änderungen nicht betroffen.

Physische DU Anwendungsprogramme müssen die interne Organisation der Daten und Zugriffs- und Speicherungsmöglichkeiten nicht kennen ⇒ sie sind von Speicher- und Zugriffstrukturänderungen nicht betroffen

Datenbank Management Systeme

DB-Typen: hierarchisch, relational, objekt-relational, objektorientiert, deduktiv, netzwerk, ...
Ein Datenbanksystem (DBS) besteht aus einem Datenbankverwaltungssystem (Software, DBMS) und einer Datenbank (Daten, DB). Anwendungsprogramme kommunizieren mit dem DBMS. Anwendungsprogramme und DBS bilden ein Informationssystem (IS).

Aufgaben DBMS

Syntaxprüfung, Objekte und Zugriffsrechte prüfen, Metadaten lesen, Zugriffsmodul generieren, Transaktionskontrolle, Daten lesen, Daten zusammenstellen, Daten ausgeben.

ANSI-SPARC-3-Ebenen-Architektur (1975)

Extern Sicht einzelner Anwendungen oder Benutzergruppen

Konzeptionell Logische Gesamtsicht

Intern Speicherung, Datenorganisation, Zugriffsstrukturen

Aufbau und Betrieb

Auf- und Ausbau bezeichnet das (iterative) erstellen verschiedener Datenbank Schemas, häufig im laufenden Betrieb. Betrieb, Nutzung und Verwaltung (Anzeigen, Einfügen, Ändern, Löschen, Backup, Restore) mit Hilfe der Daten-Manipulationssprache (DML)

Relationenmodell (E. F. Codd (1970))

Domäne Auch Wertebereich oder Datentyp: definiert die zulässigen Werte (Ganze Zahlen, Zeichenketten, ...) als Mengen und sind nicht weiter strukturiert.

Attribut besteht aus zwei Teilen: einem Namen und einer Domäne. Attribute können den Wert `NULL` haben.

Tupel bezeichnet ein atomares Datenelement aus gewerteten zusammengehörenden Attributen

Relation besteht aus zwei Teilen:

1. Schema, Heading, Intension: Name, Namen und Typen der Attribute
2. Instanz, Extension: Zeilen (n-Tupel von Werten)

Grundsätzlich ist eine Relation R eine Teilmenge des kartesischen Produkts verschiedener Mengen (hier der Wertebereiche). $R \subseteq W_1, \times W_2 \cdots \times W_n$. Dargestellt als $R(A_1, A_2, \ldots, A_n)$ (A_i steht für Attribut i). Relationen sind äquivalent ($R_1 \sim R_2$) wenn sie die gleichen Attribute enthalten. Relationen sind dublikatfrei, ungeordnet, endlich und in der ersten Normalform (alle Attribute haben atomare Typen). Als Tabelle: Spalten entsprechen Attributen, die Anzahl der Spalten entspricht dem Grad, die Zeilen entsprechen Tupeln und der Anzahl der Zeilen entspricht der Kardinalität.

Beziehung Eine Beziehung zwischen Relationen werden über Attribute und ihre Werte hergestellt.

Schlüssel identifizieren eindeutig Tupel einer Relation. Sie sind ein Attribut oder eine Attributskombination, die jedes Tupel eindeutig identifiziert und deren Wert sich während der Existenz des Tupels nicht ändert.

Relationale Algebra

Vereinigungskompatibel:

1. Gleiche Anzahl Attribute (gleiche Namen nicht notwendig)
2. Übereinstimmende Attribute haben den gleichen Typ (Domäne).

Vereinigung ($R_1 \cup R_2$)

ergibt die Menge aller Tupel, die in R_1 oder R_2 vorkommen (ohne doppelte). Voraussetzung: R_1 und R_2 sind vereinigungskompatibel. Es gilt das Kommuntativ-Gesetz: $R_1 \cup R_2 = R_2 \cup R1$

Durchschnitt ($R_1 \cap R_2$)

ergibt die Menge aller Tupel, die in R_1 und R_2 vorkommen (ohne doppelte). Voraussetzung: R_1 und R_2 sind vereinigungskompatibel. Es gilt das Kommuntativ-Gesetz: $R_1 \cap R_2 = R_2 \cap R1$

Differenz ($R_1 - R_2$ bzw. $R_1 \setminus R_2$)

ergibt die Menge aller Tupel, die in R_1 und nicht in R_2 vorkommen. Voraussetzung: R_1 und R_2 sind vereinigungskompatibel. Das Kommuntativ-Gesetz gilt nicht.

Produkt ($R_1 \times R_2$)

ergibt die Menge aller Tupelkombinationen beider Relationen (ohne doppelte). Das Kommuntativ-Gesetz gilt nicht, aber $R_1 \times R_2 \sim R_2 \times R_1$. (Es gäbe auch noch die praktisch bedeutungslose Division).

Selektion ($\sigma_P(R)$)

P steht das Prädikat, die Selektionsbedingung. Die Selektion ergibt die Menge aller Tupel aus R für die P gilt. Die Selektion hat nichts mit der Reihenfolge (Sortierung) zu tun.

Projektion ($\pi_L(R)$)

L steht für die Attributkombination aus R. Die Projekt ergibt die Menger aller Attribute aus R, die L sind. Theoretisch werden doppelte eliminiert.

Verbund, Join ($R_1 \bowtie R_2$)

ergibt die Verkettung der Attribute von R_1 mit den von R_2 bei denen gemeinsame, vereinigungskompatible Attribute gleich sind. Verknüpfbar sind zwei Tupel, wenn sie in allen gemeinsamen Attributen übereinstimmen.

Natural, Auto Join	$\sigma_{R_1.s_1=R_2.s_1 \wedge R_1.s_2=R_2.s_2 \ldots}$
Equi-Join	Prüfung nur auf Gleichheit. Attribute mit logischem Und verknüpft.
Theta Join ($L \bowtie_\theta R$)	Kartesisches Produkt mit Selektion (bsp. $R_1.a_1 > R_2.a_2$)
Cross Join	keine gemeinsamen Attribute (kartesisches Produkt)
Semi Join ($L \ltimes R$)	Natural Join, nur Attribute von L
Left outer join ($L \rtimes R$)	Alle von Tupel von L mit `NULL` für Attribute von R, die nicht verknüpfbar sind.
Right outer join ($L \ltimes R$)	Alle von Tupel von R mit `NULL` für Attribute von L, die nicht verknüpfbar sind.
Full outer join ($L \rtimes R$)	Alle von Tupel von L und R mit `NULL` für Attribute, die nicht verknüpfbar sind.

Funktionale Abhängigkeit

Das Attribut bzw. die Attributskombination Y ist funktional abhängig vom Attribut bzw. von der Attributskombination X derselben Relation R, wenn zu einem bestimmten Wert von X höchstens ein Wert von Y möglich ist $(X \to Y)$

Regeln

Projektivität	$(\{Y\} \subseteq \{X\}) \Rightarrow (X \to Y)$
Distributivität	$(X \to YZ) \Rightarrow (X \to Y, X \to Z)$
Additivität	$(X \to Y, X \to Z) \Rightarrow (X \to YZ)$
Transitivität	$(X \to Y, Y \to Z) \Rightarrow (X \to Z)$
Pseudotransitivität	$(X \to Y, YW \to Z) \Rightarrow (XW \to Z)$
Akkumulation	$(X \to YZ, Z \to W) \Rightarrow (X \to YZW)$
Erweiterung	$(X \to Y, W \to Z) \Rightarrow (XW \to YZ)$

Datenbank-Entwicklung

Anforderungsanalyse, Spezifikation	Datenanforderungen durch Interviews, Fragebögen ermitteln
Konzeptioneller Entwurf	Konzeptionelles Model beschreiben (DBMS unabhängig) ⇒ Konzeptionelles Schema
Logischer Entwurf	Logisches Schema aus dem Datenmodell erstellen, optimieren, externe Schemas definieren (DBMS abhängig) ⇒ Logisches Schema
Physischer Entwurf	Internes Schema definieren (DBMS abhängig) ⇒ Physisches Schema
Deklaration Schemas	In der DDL des DBS

Kriterien: Vollständigkeit, Korrektheit, minimale Redundanzen, Lesbarkeit, Erweiterbarkeit, Normalisierung.
Refactoring eines Modelles ist sehr aufwändig und wird deswegen kaum gemacht. Daher am besten mit einem kleinen Kernteam (Business Spezialist (Anforderungen/Konzept), Datenmodellierer(Konzept/Logik), Datenbank Designer(Logik/Physisches Model)) arbeiten, das Entwürfe macht, die von einem grösseren Team (Datenbankspezialisten, Projektmanager, Auftraggeber) evaluiert werden.

Konzeptionelles Modell weitgehend technologie-unabhängige Spezifikation der Daten, die in der Datenbank gespeichert werden sollen

Logisches Modell Übersetzung des konzeptionellen Schemas in Strukturen, die mit einem konkreten DBMS implementiert werden können

Physisches Modell alle Anpassungen, die nötig sind um eine befriedigende Leistung im Betrieb zu erreichen (Datenverteilung, Indexierung, ...)

Datenmodellierung

Daten- und funktionale Anforderungen definieren. Anschliessend auf die Daten Abstraktionskonzepte anwenden: Klassifikation (gleiche oder ähnliche „Dinge“), Aggregation (Zusammenfassen von „Dingen“, Name, Strasse, ... zu Adresse), Generalisierung und Spezialisierung (Verallgemeinerung, Teilmengenbeziehungen, Zerlegung in disjunkte Untermengen).
Bei der Modellierung gilt es sich auf den Anwendungsbereich zu beschränken. Die Modellierung muss unabhängig von der späteren Implementierung sein.

Entity-Relationship-Model (P. P. S. Chen, 1976)

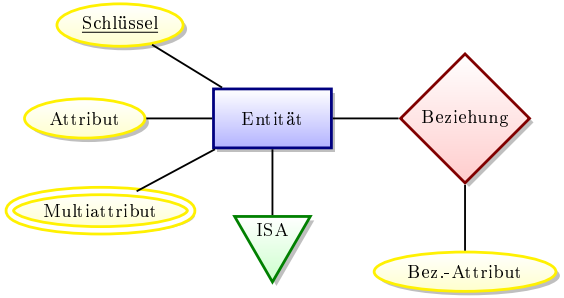
Entität Ein konkretes oder abstraktes Objekt der realen oder einer Vorstellungswelt, welches eindeutig identifiziert werden kann. Eine Entität hat Attribute (Bsp. Herr Aebi).

Entitätsmenge Gleichartige Entitäten, d.h. Entitäten mit denselben Eigenschaften (Bsp. Personen).

Identifikationsschlüssel Ein Attribut oder eine Menge von Attributen, welche eine Entität eines Entitätstyps innerhalb dieses Typs eindeutig identifiziert (Bsp. Personalnummer)

Beziehung Eine Beziehung zwischen Entitäten $e_1, e_2, \ldots, e_n (n \geq 2)$ ist ein n- Tupel, wobei jedes e_i eine Entität eines Entitätstyps $E_i (1 \leq i \leq n)$ ist. Wenn derselbe Entitätstyp mehrfach vorkommt (E_i und E_j mit $i \neq j$), dann müssen deren verschiedene Rollen mit Hilfe zusätzlicher Rollennamen bezeichnet werden.
Gleichartige Beziehungen werden zu einem Beziehungstyp zusammengefasst (Bsp. Person als Autor eines Buches)
Identifikationsattribute einer Entität, die an einer Beziehung beteiligt ist, gehört implizit zur Beziehung. Eine Beziehung kann darüber hinaus weitere Attribute haben. Der Identifikationschlüssel einer Beziehung wird aus Identifikationsattributen, der an der Beziehung beteiligten Entitätstypen gebildet.

Generalisierung (is-a) Eine Generalisierung ist ein Enitätstyp G der die gemeinsamen Attribute der spezialisierten Entitäten $E_1, E_2, \ldots, E_n$ umfasst. Eine Spezialisierung ist disjunkt, wenn $\bigcap(E_1, E_2, \ldots, E_n) = \emptyset$. Ansonsten ist er überdeckend.



Beziehungstypen (Kardinalität)

1:1 Jedem Element aus E_1 ist höchstens 1 Element aus E_2 zugeordnet

1:M Jedem Element aus E_1 können beliebig viele Elementa aus E_2 zugeordnet sein, jedem Element aus E_2 aber höchstens ein Element aus E_1

M:M Jedem Element aus E_1 können beliebig viele Elemente aus E_2 zugeordnet sein und umgekehrt. Diese Beziehungen müssen für Datenbanken aufgelöst werden.

Attribute werden aus Gründen der Übersichtlichkeit im ERM oft weggelassen und in ergänzenden Dokumenten festgehalten. Die Details entstehen in der Regel beim logischen Entwurf. Der Entwurf ist ein iterativer Prozess an dem mehrere Personen beteiligt sein sollten.

Logischer Entwurf

Grundsätzlich wird aus einer Entität eine Relation, ebenso wird aus einer Beziehung eine Relation. Mehrwertige Attribute werden aufgeteilt (zusätzliche Relationen), M:M Beziehungen werden mit Zwischentabellen aufgelöst (zusätzliche Relationen).

Am Ende steht ein logisches Schema, das in SQL (DDL) formuliert ist.

Normalisierung

Unsachgemässe Entwürfen können zu Mutationsanomalien (unerwünschte Auswirkung von Änderungen) führen. Durch Normalisierung wird dies verhindert.

Anwendungswissen für die Normalisierung: Repetitionsgruppen (was kommt wie oft vor?), Schlüssel, funktionale Abhängigkeiten, Codierungen.

1. **Normalform** Alle Attribute enthalten nur atomare Werte (mehrwertige Attribute werden in Tupel umgewandelt)
2. **Normalform** Eine Relation ist in der zweiten Normalform, wenn die erste Normalform vorliegt und jedes Nichtschlüsselattribut vom Primärschlüssel voll funktional abhängig ist. (Mehrere Tabellen entstehen)
3. **Normalform** Die dritte Normalform ist erreicht, wenn sich das Relationenschema in 2NF befindet, und jedes Nichtschlüsselattribut von keinem Schlüsselattribut transitiv abhängt.

1NF → 2NF

1. Bestimme alle Nichtschlüsselmerkmale, die bereits von einem Teilschlüssel funktional abhängig sind.
2. Bilde aus den Teilschlüsseln und allen von ihnen funktional abhängigen Nichtschlüsselmerkmalen eigene Tabellen.
3. Entferne aus der ursprünglichen Tabelle alle nicht voll funktional abhängigen Nichtschlüsselmerkmale.

2NF → 3NF

1. Bestimme alle vom Schlüssel transitiv abhängigen Nichtschlüsselmerkmale.
2. Bilde aus diesen transitiv abhängigen Nichtschlüsselmerkmalen und den Nichtschlüsselmerkmalen, von denen sie funktional abhängig sind, eigene Tabellen.
3. Entferne aus der ursprünglichen Tabelle alle transitiv abhängigen Nichtschlüsselmerkmale.

Durch Normalisierung werden Redundanzen vermindert und Anomalien vermieden. Allerdings entstehen so viele kleine Relationen, die einen hohen Verarbeitungsaufwand nach sich ziehen (Update-Query-Tradeoff).

Physischer Entwurf

Der Vorgang ist iterativ. Tuning erfolgt während des laufenden Betriebs.

- Speichermedium wählen (RAID 0, 1, 5)
- Zugriffsart
- Block-, Seitenzuweisung, Verteilung der Daten
- Indexierung
- Monitoring
- Denormalisierung

Speicherhierarchie

Je schneller, desto kleiner, desto teurer ...

1. Cache Speicher
2. Arbeits-, Programmspeicher
3. Massenspeicher, Festplatte, USB-Stick
4. Archiv-Speicher (Magnetbänder, optische Platten)

Festplatten

bestehen aus einem Plattenstapel. Eine Platte ist in Spuren (Tracks), diese in Sektoren unterteilt. Übereinanderliegende Spuren heissen Zylinder. Mehrere Spuren bilden einen Block.
Daten werden blockweise geschrieben/gelesen. Es werden immer ganze Blöcke übertragen.
Das DBMS stellt einen Disk Space Manager (verwaltet Zugriff auf Sekundärspeicher) und einen Buffer Manager (verwaltet den Hauptspeicher) zur Verfügung. Das Betriebssystem gewährt mit dem Platten-Controller den Zugriff.

Datenorganisation

Unsortierte Daten Sequentielle Suche ($\frac{n}{2}$ Zugriffe)
Sortierte Daten Binäre Suche ($\log_2 n$ Zugriffe)
n = Anzahl Datensätze. Sortierung nur für ein Kriterium möglich.
Indexe sind verschiedene sortierte Verzeichnisse. Aber: Indexe brauchen zusätzlichen Speicherplatz und müssen gepflegt werden (Update-Query-Tradeoff) Indexe sind das primäre Optimierungsmittel!

- \+ Primärschlüssel (i. d. R. automatisch)
- \+ Fremdschlüssel (beschleunigt Joins)
- \+ Attribute mit häufigen Abfragen und kleinen Resultatmengen
- \+ Attribute, die oft sortiert ausgegeben werden
- − Attribute, deren Wert sich häufig ändert
- − Grosse Attribute, komplexe Attributkombinationen

Datenorganisationsformen

Heap-Datei Bei einer Heap-Datei werden die Datensätze unsortiert hintereinander (sequentiell) geschrieben. Einfach, aber nur sequentielle Suche ist möglich (Suchen, bis gefunden).

Hash-Verfahren Dabei werden die Datensätze unterschiedlichen Bereichen zugeordnet. Innerhalb der Bereiche sind die Datensätze wieder ungeordnet. Die Zuordnung der Datensätze zu den Bereichen erfolgt über eine Hashfunktion. Einfach, aber einzelne Bereiche müssen sequentiell durchsucht werden.

ISAM-Verfahren Beim ISAM-Verfahren (Index Sequential Access Method) wird neben der eigentlichen Datendatei, die die Datensätze in sortierter Reihenfolge enthält, eine weitere Datei, die so genannte Index-Datei gepflegt. Diese Index-Datei ermöglicht einen beschleunigten Zugriff auf die Daten. Suchen ist sehr schnell, aber die Index-Datei muss ständig mit angepasst werden.

B*-Baum-Verfahren Betrachtet man die Indexdateien des ISAM-Verfahrens wiederum als Datensätze, so kann zu diesen Datensätzen wiederum eine Indexdatei generiert werden. Durch diese Verschachtelung von Indexdateien kommt man zu einer baumartigen Struktur. Merkmale eines B*-Baumes:

- Jeder Knoten des Baumes kann mehrere Kindknoten haben (Mehrwegbaum)
- Der Baum ist ausgeglichen (balancierter Baum)
- Die Knoten enthalten nur Verweise und nicht die eigentlichen Daten

Es gibt so zwar keine grossen Index-Dateien, doch einfügen und löschen ist kompliziert. Die optimale Variante für überwiegend lesenden Zugriff.

Sekundärindex Index über ein zusätzliches Merkmal. Lesender Zugriff auf zusätzliches Merkmal ist sehr schnell, aber einfügen und löschen ist noch komplizierter als beim B*-Baum-Verfahren.

Anfrageoptimierung

Die Anzahl Diskzugriffe sind zu minimieren. Dabei werden Abfragebäume visualisiert und mit Hilfe der relationalen Algebra optimiert (Kommutativität, Relationen mit höherer Kardinalität in die innere Schleife (Nested-Loop-Join), sortieren vor dem mischen (Sort-Merge-Join), mit Selektion und Projektion Zwischenergebnisse verkleinern.
Die Optimierung hängt oft vom konkreten Datenvolumen ab.

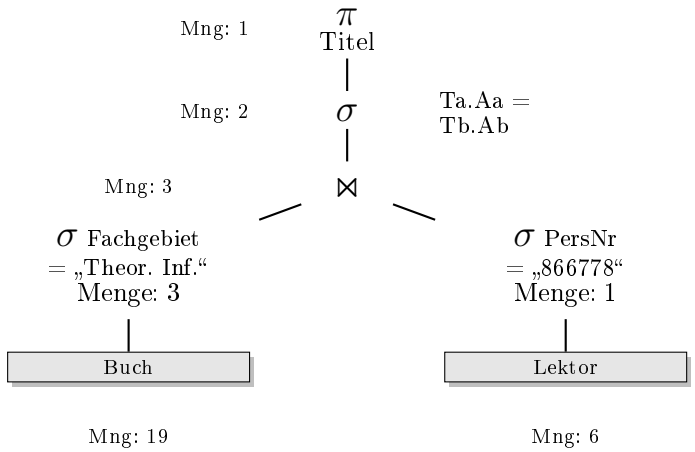
Regeln

- Verbund ($\bowtie$), Vereinigung ($\cup$), Durchschnitt ($\cap$) und Produkt ($\times$) sind kommutativ und assoziativ.
- Selektionen sind vertauschbar: $\sigma_P(\sigma_Q(R)) = \sigma_Q(\sigma_P(R))$
- Konjunktionen in einer Selektionsbedingung kann in mehrere Selektionen aufgebrochen werden: $\sigma_{P_1}(R) \wedge \sigma_{P_2}(R) \wedge \cdots \wedge \sigma_{P_n}(R) = \sigma_{P_1}(\sigma_{P_2}(\ldots(\sigma_{P_n}(R))))$
- Projektion mit Vereinigung vertauschbar: $\pi_L(A \cup B) = \pi_L(A) \cup \pi_L(B)$
- Disjunktion in Konjunktion umwandelbar: $\neg(A \wedge B) = \neg A \vee \neg B, \neg(A \vee B) = \neg A \wedge \neg B$

Das DBMS nutzt diese Regeln.

- Wenn möglich vor dem Join sortieren
- Wenn Nested-Loop-Join notwendig, vor dem Join einschränken (σ, π)
- Indexe reduzieren den Aufwand, kosten aber Aufwand ($n \log_n$)).

Optimierter Abfragebaum



Anfragebeispiel: $\pi_{t1.f1,t2.f1}(\sigma_{t1.f1=\text{„kriterium“}}(t1 \bowtie t2))$

SQL (IBM 1974)

Zwar gibt es einen Standard, doch eingesetzte Systeme sind proprietär. Einerseits aus kommerziellen, andererseits aus Trägheitsgründen - das SQL-Komitee hingt den Bedürfnissen des Marktes hinterher.
Leider ist SQL für „reale“ Anwendungen zu wenig mächtig. (Keine Benutzerschnittstellen, komplexe Berechnungen nicht möglich, Zugriff auf Geräte nicht möglich, Vernetzung problematisch) ⇒ SQL wird mit Programmiersprachen (Java/JPA) kombiniert.

Data Definition Language (DDL)

Erzeugen und Ändern von Datenbank-Objekten.
Als erstes Datenbanken erzeugen:

```
CREATE DATABASE IF NOT EXISTS database
```

Datentypen

- Zahlen (Ganze (`integer, bigint`), Fix- (`decimal(p,s), numeric(p,s)`) und Fliesskomma (`float(p), real(p)`))
- Zeichenketten (`char(n), varchar(n)`)
- Kalenderdaten (`date, time, datetime`)
- Sonstige (herstellerabhängig)

```
CREATE TABLE Products(
    PNo INTEGER PRIMARY KEY,
    Descr CHAR(30) NOT NULL UNIQUE,
    Weight FLOAT CHECK(Weight > 0.0),
    ...
);
CREATE TABLE Orders(
    OrdNo INTEGER,
    ...,
    PNo INTEGER NOT NULL REFERENCES Products(PNo),
```

```
    CNo INTEGER NOT NULL REFERENCES Customers(CNo)
        ,
    Status CHAR(7) DEFAULT 'ordered',
    ValidDate DATETIME NOT NULL,
    CONSTRAINT PK_Orders PRIMARY KEY(OrdNo),
    UNIQUE (ValidDate,PNo,CNo)
);
ALTER TABLE table ADD COLUMN newColumn datatype;
DROP TABLE table;
```

Data Manipulation Language (DML)

Insert

```
INSERT INTO table (columnA, columnB) VALUES ("
    value1", "value2");
INSERT INTO table (columnA, columnB)
    SELECT field1, field2 FROM tableB WHERE ...;
```

Würden Integritätsbedingungen verletzt, wird nichts geändert. Fehlende Attribute werden mit dem Default Wert oder `NULL` gefüllt.

Update

```
UPDATE table SET field = "value" WHERE ...;
UPDATE table SET field = field + 1;
```

Würden Integritätsbedingungen verletzt, wird nichts geändert.

Delete

```
DELETE FROM table WHERE ...;
```

Würden Integritätsbedingungen verletzt, wird nichts geändert.

Select

```
SELECT * | [ALL | DISTINCT] expression [[AS]
    column-alias][,...]
FROM table [[AS] alias]
[,|CROSS JOIN table [[AS] alias] |
    NATURAL [INNER|OUTER]{LEFT|RIGHT|FULL}]
    JOIN table [[AS] alias]
     |[INNER|OUTER]{LEFT|RIGHT|FULL}]
    JOIN table [[AS] alias]
    {ON condition | USING (col-list)}
|UNION join-table [[AS] alias] [,...]]
[WHERE col-condition]
[GROUP BY colname [,...]
[HAVING group-condition]]
[UNION|INTERSECT|EXCEPT ...]
[ORDER BY col [ASC|DESC][,...]];

SELECT Name
FROM Customers AS c,
    (SELECT CNo FROM Orders
    WHERE Status <> 'paid' AND
    Validdate < DATE '2003-10-01') AS o
WHERE o.CNo = c.CNo

/* Join mit ON */
SELECT Name FROM Customers c
JOIN Orders o ON o.CNo = c.CNo
WHERE Status <> 'paid' AND Validdate < DATE '
    2003-10-01'

/* Join mit USING */
SELECT Name FROM Customers
JOIN Orders USING(CNo)
WHERE Status <> 'paid' AND Validdate < DATE '
    2003-10-01'

/* NATURAL JOIN */
SELECT Name FROM Customers
NATURAL JOIN Orders
WHERE Status <> 'paid' AND Validdate < DATE '
    2003-10-01'
```

Regeln für `NULL`

$$\neg \texttt{NULL} = \texttt{NULL}$$

$$\texttt{NULL} \vee \texttt{NULL} = \texttt{NULL} \qquad \texttt{NULL} \wedge \texttt{NULL} = \texttt{NULL}$$

$$\text{false} \vee \texttt{NULL} = \texttt{NULL} \qquad \text{false} \wedge \texttt{NULL} = \text{false}$$

$$\text{true} \vee \texttt{NULL} = \text{true} \qquad \text{true} \wedge \texttt{NULL} = \texttt{NULL}$$

Selektionsprädikate

`IN` **SELECT** * **FROM table WHERE** field **IN** ('value1', 'value2');

`LIKE` **SELECT** * **FROM table WHERE** field **LIKE** 'value%';

`BETWEEN` **SELECT** * **FROM table WHERE** field **BETWEEN lower AND upper**;

`IS NULL` **SELECT** * **FROM table WHERE** field IS **NOT NULL**;

Aggregatfunktionen: [**COUNT**|**SUM**|**AVG**|**MIN**|**MAX**](field)
`COUNT` eleminiert Duplikate nicht, zählt `NULL`-Wert mit. Die anderen Aggregatfunktionen ignorieren `NULL`-Werte.
Gruppen sind eine Multimenge von Tupeln mit denselben Werten für das Gruppierungskriterium. `HAVING` entspricht der `WHERE`-Klausel.

Data Controll Language

Rechte Vergabe (`GRANT, REVOKE`), Schutz vor Ausfällen. Job des Datenbankadministrators.

Transaktionen

Eine Transaktion ist eine Folge von Schreib- und Lese-Zugriffen, die eine geschlossene Arbeitseinheit bilden.

Multiuser

Ein DBMS simuliert gleichzeitigen Zugriff, weil nur ein Zugriff pro CPU pro Zeit möglich ist. Es behandelt Fehler automatisch und startet nach schwerwiegenden Fehlern neu.
Nutzer erhalten alternierend kurze Prozessorzeit. Dadurch laufen während des Prozesses eines Benutzers auch die Prozesse anderer Nutzer ab. So können inkonsistente Datenbestände entstehen, was das DBMS mit Transaktionen verhindert. Mögliche Probleme:

Lost-Update Überschreiben bereits getätigter Updates (ohne neuerliches Read) ⇒ Für Dauer der Transaktion sperren.

Dirty-Read Lesen von Veränderungen durch später abgebrochene Transaktionen ⇒ Nur bestätigte Transaktionen berücksichtigen.

Phantom-Read Lesen von zwischenzeitlich von anderen Transaktionen durchgeführten Veränderungen. ⇒ Nur Datenbankzustand zu Beginn einer Transaktion verwenden.

Gleichzeitige Zugriffe werden durch Ablaufpläne (Schedules) modelliert.

ACID-Prinzip

A - Atomarität - Zusammenhängende Aktionen ganz oder gar nicht ausführen.
C - Consistency - Alle Aktionen hinterlassen die DB in einem konsistenten Zustand.
I - Isolation - Transaktionen laufen unabhängig von einander ab.
D - Durability - Resultate bleiben nach Transaktion persistent.
Commit - Transaktion durchführen, Rollback = Abbruch:

```
BEGIN TRANSACTION
COMMIT TRANSACTION
ROLLBACK TRANSACTION
```

Transaktionsverarbeitung

Datenobjekte werden gesperrt. Problem: Deadlock - gegenseitiges Warten auf Freigabe von Sperren. Solche Zyklen werden vom DBMS erkannt.

Integritätssicherung

Sicherstellen, dass nur „richtige“ Daten in die DB gelangen. Die Menge der konsistenten Zustände wird eingeschränkt.
Integritätsicherung ist eine zentrale Aufgabe eines DBMS.

Möglichkeiten

- Attribut-Bedingungen (Datentyp, `NOT NULL, CHECK`)
- Wertebereichsbedingungen (`CREATE DOMAIN ...AS ...CHECK ()`)
- Tupel-Bedingungen (`CHECK`)
- Relationen-Bedingungen (`PRIMARY KEY`)
- Referenzielle Bedingungen (`FOREIGN KEY ...REFERENCES ...`)

Einschränkungen können auch verzögert aktiviert werden:

```
CREATE TABLE ... CONSTRAINT field ... INITIALLY {
    DEFERRED|IMMEDIATE}
```


Revision: 1.0, Datum: 23. Juni 2013