DEFINITIONS and THEORY

Access Path: the algorithm and data structure used for retrieving and storing data in a table, affects the execution time of the SQL stmt

Aggregation: A feature of the entity relationship model that allows a relationship set to participate in another relationship set.

Candidate key: (or just 'key') Removing any of its attributes would make it no longer a superkey

Concurrency control*:* The protocol that manages simultaneous operations against a database so that serialisability is assured

Covering index: index that contains all attributes for a SQL query

Artificial Keys: Keys that are introduced and have no external meaning

Data Warehousing: Consolidate data from many sources in one large repository

DDL: Data Definition language(add/delete/update records)

DML: Data Manipulation language(define tables and other database objects)

DTD: Document Type Definition DCL: Data Control language(set access privileges for users)

Deadlock: cycle of transactions waiting for locks to be released by each other

SQL: Structured Query Language
Based on formal query languages: Relational Algebra and Relational Calculus

Decomposition**:** replacing R by two or more relations such that each new relation contains a subset of the attributes of R, and every attribute of R appears in one of the new relations and all the new relations differ

Drilling down**:** executing a series of queries moves down a hierarchy

Dynamic IC: Are predicates on database state changes

ETL process: Extract, Transform, Load

Entity: a distinguishable object about which you want to gather and store data.

Entity Type (entity set): a collection of entities that share common properties or characteristics

Foreign Key: identifiers that enable a dependent relation to refer to its parent relation

Functional dependency: The value of one attribute determines the value of another attribute

Indexes: An access path to efficiently locate rows via search key fields without having to scan the entire table.

Integrity constraints**:** A condition that must be true for any instance

Key**:** minimal set of attributes that uniquely identifies an entity

Normal forms: A set of restrictions that determine a table's degree of immunity against logical inconsistencies and anomalies

Natural Keys: Keys represent conceptual uniqueness constraints external to the database

Primary index: index whose search key specifies the sequential order of the file

Primary Key: a unique identifier in a relation (a key unique for each record.)

Referential integrity**:** for each tuple in the referring relation whose foreign key value is A, there must be a tuple in the referred relation with a candidate key that also has the value A

Relational Algebra**:** defines some basic operators that can be used to express a calculation

Relational views: a virtual relation that stores a definition rather than a set of tuples

Surrogate key:If an artificial key is used as primary key when a natural key also exists, we can say the artificial key is a surrogate key for the natural key

Schema normalisation: process of validation and improving a logical design so that it satisfies certain constraints and avoids duplication

Secondary index: index whose structure is separated from the data file and search key is not in sequential order

Serialisability: A sequence of database operations is serialisable if it is equivalent to a serial execution of the involved transactions

Star Schema: fact and dimensions relations displayed in an ERD

Static IC: Describe conditions that every legal instance of a database

Stored Procedures: run application logic within the database server

Semi-structured data: Self describing irregular data no prior structure

Superkey**:** a combination of columns that uniquely identifies any row
Strict Superkey: have at least one of its members removed and still be a superkey

Snapshot Isolation: the database state produced by the execution of the first transactions to commit

Weak entity type: An entity type that does not have a primary key. The discriminator (or partial key) of a weak entity type is the set of attributes that distinguishes among all the entities of a weak entity type related to the same owning entity.

SQL/XML: supports storing and export of data as XML

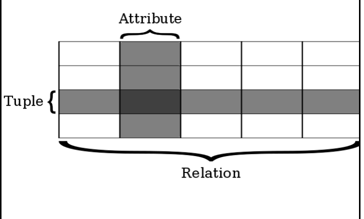
Anomaly: problems that arise in the data due to a flaw in the design

Union	(∪)	tuples in relation 1 or in relation 2.
Intersection	(∩)	tuples in relation 1, as well as in relation 2.
Difference	(-)	tuples in relation 1, but not in relation 2.
Rename	(ρ)	allows us to rename a field or relation.
Cross-product	(×)	to combine *every* tuple from two relations.
Join	(⋈)	to combine *matching* tuples from two relations.

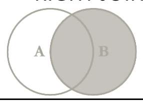
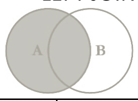
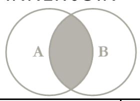
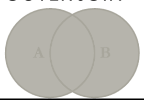
RA expression: $\pi_{title}(\sigma_{points=6}(UnitOfStudy))$

SQL query:
```
SELECT title
FROM UnitOfStudy
WHERE points=6;
```

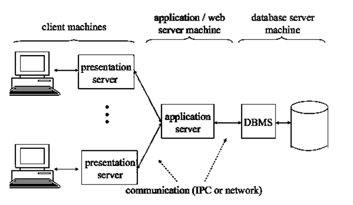
Overlap Constraints (重叠)(similar to key constraint)
Default Overlapping: an entity can belong to more than one lower-level entity set
Disjoint: an entity can belong to only one lower-level entity set
Covering Constraints (similar to Participation Constraint)
Default Partial: an entity need not belong to one of the lower-level entity sets
Total: (superclass- thick line -isa)an entity must belong to one of the lower-level entity sets

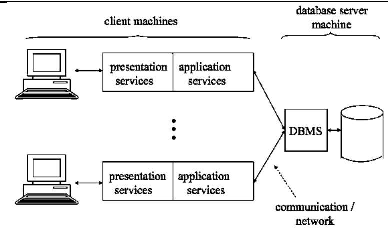


Integrity constraints
NOT NULL, DEFAULT, NULL
UNIQUE
CONSTRAINT … PRIMARY KEY
"" FOREIGN KEY … REFERENCES
ON DELETE ON UPDATE CASCADE
CHECK(… IN ("", ""))

RIGHT JOIN	LEFT JOIN	INNER JOIN	OUTER JOIN
			

1NF	2NF	3NF
break data remove duplicates	no partial dependency on key	get rid of dependent fields e.g. average





BCNF algorithm
- ⊗ pick a relation that doesn't have a key to the left
- ⊗ decompose it into two relations
  - ○ R1(A,B)
  - ○ R2(A, rest)
- ⊗ repeat until no relations violate BCNF
- ⊗ check that the join will produce the same output

Null Value
PRO: ordinary values don't work
CON: complication

Preventing SQL Injection Attacks
check parameters
use dynamic SQL statements
restrict privileges
never directly return errors

Limitations of indexes
addition I/O to access pages
must be updated when tables are modified
decision on good indexes is hard

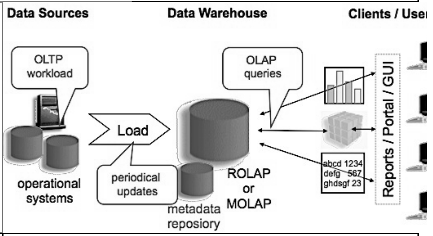
Requested \ Held	Shared	Exclusive
Shared	OK	T2 wait on T1
Exclusive	T2 wait on T1	T2 wait on T1

Isolation levels
READ UNCOMMITTED
READ COMMITTED
REPEATABLE READ
SERIALIZABLE

Stored Procedure +ves
central code base
improved maintainability
additional abstraction layer
reduced data transfer
DBMS-centric security
consistent logging/auditing

Data Warehousing -ves
Semantic Integration – eliminating mismatches from multiple sources
Heterogeneous sources – variety of source formats, *Load, Refresh, Purge*
Metadata Mgmt – track source loading time

Choosing an index
B+-Tree **-** point and range queries, prefix searches
Hash **-** equality searches
If there is a PK, no clustered
Benefit most queries
sequential PI, otherwise SI



OLTP: Maintains a database that is an accurate model of some real-world enterprise
- short simple transactions
- frequent updates

OLAP: Uses information in the database to guide strategic decisions
- complex large transactions
- infrequent updates
- no need for up-to-date data

DTD
- Grammar
- Elements + Attributes
- Only 'Part of' relationships
- part of prolog of XML doc

XMLSchema
- Structure and typing
- Elements, attributes, simple and complex types and groups
- attribute of the doc elements

CODE

Participation / Cardinality

at most one $\longrightarrow$ 0…*

at least one $\boldsymbol{—}$ 1…1

exactly one $\boldsymbol{\longrightarrow}$ from E …. to E

Operator	Meaning	Example
=	equal to	WHERE age = 20
>	greater than	WHERE age > 20
>=	greater than or equal	WHERE age >= 20
<	smaller than	WHERE age < 25
<=	smaller than or equal	WHERE age <= 25
BETWEEN	between start and end value (inclusive)	WHERE age BETWEEN 20 and 25
IN	set of valid values	WHERE age IN (20,21,22)

Not Equal To '<>' or '!='

Negation with NOT

For a filter condition at the WHERE clause, you can combine multiple conditions using conjunctions (AND) and disjunctions (OR).

Set Operations in SQL

UNION	Set union $(r_1 \cup r_2)$	(select…from…)union
INTERSECT	Set intersection $(r_1 \cap r_2)$	(select…from…)
EXCEPT	Set difference $(r_1 \setminus r_2)$. Note that in some SQL dialects, notably Oracle, this can be called MINUS.	

Distinct e.g. COUNT(DISTINCT name)

Assertion

CREATE ASSERTION … CHECK (NOT EXISTS (<query don't want>))

CREATE ASSERTION … CHECK (EXISTS (<query want>))

Trigger

CREATE TRIGGER
BEFORE/AFTER
INSERT/DELETE/UPDATE ON …
FOR EACH ROW
WHEN (…)
EXECUTE PROCEDURE ….();

Time

SELECT/WHERE(EXTRACT(day/…/second from CURRENT_TIMESTAMP/TIME/DATE)

Views

Although a view is queried like a table, there is nothing stored in it. The contents are dynamically retrieved from the underlying tables each time it is referenced in a query.

CREATE VIEW name AS <query expression>

A view on the `Student(sid,name,birthdate)` showing their age.

```
CREATE VIEW ageStudents AS
    SELECT sid, name, extract(year from sysdate) -
                         extract(year from birthdate) AS age
      FROM Student
```

A view on the female students enrolled in 2012sem1 using `Enrolled(sid, uos, semester, grade)`

```
CREATE  VIEW FemaleStudents2012 (name, grade)
             AS SELECT S.name, E.grade
                  FROM Student S, Enrolled E
                 WHERE S.sid = E.sid  AND
                       S.gender = 'F' AND
                       E.semester = '2012sem1'
```

$\underbrace{\gamma_{\{aggregation\ ops\}}}_{\text{aggregation operation = func(attribute)}}(relation)$ *Aggregation functions* operate on the multiset of values of a column of a relation, and return a value

- **AVG**: average value
- **MIN**: minimum value
- **MAX**: maximum value
- **SUM**: sum of values
- **COUNT**: number of values
- **COUNT(*)**: count of rows

List entries

no tuples SELECT … FROM … WHERE NOT EXISTS (SELECT 1 FROM … WHERE…)

atleast x ORDER BY x

Nested subqueries

WHERE clause

SELECT… FROM … WHERE NOT EXISTS/ EXISTS/IN/UNIQUE (<query>)

FROM clause

SELECT … FROM (<query>) WHERE…

HAVING clause

SELECT … FROM … WHERE … HAVING … comparison_operator (<query>)

Group/Order

```
SELECT category_id, name, COUNT(film_id) AS count
  FROM Category LEFT OUTER JOIN Film_Category
                         USING (category_id)
 GROUP BY category_id, name
 ORDER BY count DESC, name ASC;
```

IS NULL and IS NOT NULL `\pset null '[NULL]'`

`COALESCE(grade, '[UNKNOWN]')`

Other manipulations

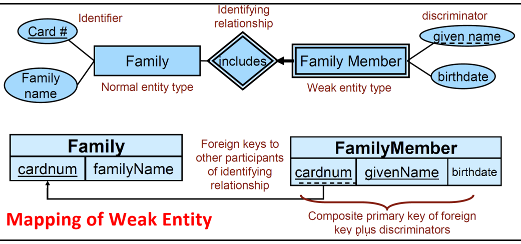
change null	COALESCE(…, …)
limit out put	LIMIT x;
casting	CAST(… AS TYPE)

String operations

SELECT … FROM … WHERE … LIKE '%...%'
(*starting with …% ending with %…*)
Like '… _ _ _'
Lower()/Upper()
Function: **Similar to**
e.g. SIMILAR TO 'COMP[[:digit:]]{4}'
e.g. SIMILAR TO '(Advanced|Data)%'; start with A…/D…

Cases

SELECT … CASE WHEN … = '…' THEN '….' END FROM … WHERE …



Mapping of Weak Entity

```
SELECT *
  FROM Student NATURAL JOIN Enrolled
 WHERE uosCode = 'INFO2120';
```

r1 (INNER)JOIN r2 ON (condition)
r1 (INNER)JOIN r2 USING (field(s))

r1 LEFT OUTER JOIN r2
r1 RIGHT OUTER JOIN r2
r1 FULL OUTER JOIN r2
r1 CROSS JOIN r2 ---contain all tuples of two given relations with the cartesian product

Transaction:

a collection of one or more operations on one or more databases, which reflects a discrete unit of work

Transaction ACID:

Atomicity
Transactions cannot be partially committed, they either happen completely
or not at all

Consistency
A transaction takes the database from one valid state to another by
some semantically meaningful transition

Isolation
The changes made by one transaction should not be visible to another.

Durability
After a system failure the database is guaranteed to be restored to the last
consistent state (i.e., as it was immediately after the last successful COMMIT).

Stored Procedure

CREATE PROCEDURE (…, …)
CREATE FUNCTION …(text)
RETURNS TABLE (column DATATYPE, …)
AS $$
BEGIN
RETURN QUERY (<query>);
END; $$
LANGUAGE PLPGSQL STABLE EXTERNAL SECURITY DEFINER;

Cube Operator/Rollup

SELECT y, x, ROUND(AVG(z), 0)
FROM …., …, … WHERE ….
GROUP BY CUBE/ROLLUP (y, x);

DTD

<!ELEMENT book (title,author+,price?)>
<!ELEMENT title (#PCDATA)>

Deferring Constraints

NOT DEFERRABLE – *checked immediately*
INITIALLY DEFERRABLE – *wait til end*
NITIALLY IMMEDIATE – *chck nw change later*

Transactions

BEGIN TRANSACTION
COMMIT
ROLLBACK

HAVING clause Limit number限制个数

SELECT AVG(mark) FROM … GROUP BY … HAVING COUNT(sid) > (SELECT COUNT(sid) FROM … WHERE uos_code='INFO2120')

FROM clause

SELECT … FROM (SELECT sid FROM … JOIN Enrolled USING (sid)) WHERE grade = 'HD'

Unique clause Exists/in

Find students who never repeated any subjects.

```
SELECT sid, name
  FROM Student
 WHERE UNIQUE( SELECT uos_code
                 FROM Enrolled
                WHERE Enrolled.sid = Student.sid )
```

Data Type

SMALLINT — 2 byte numeric integer value
INTEGER — 4 byte numeric integer value
FLOAT — 8 byte floating point value
CHAR(n) — fixed-length string of n characters
VARCHAR(n) — variable-length string of 0 to n characters

Data type	Description	Example value
DATE	A date	'2011-04-03'
TIME	A time	'19:14:06.977434+11'
TIMESTAMP	A date and time	'2011-04-03 19:14:33.974799+11'

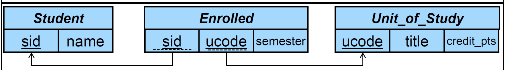
Insert/Update/Delete

Insertion of new data into a table / relation
Syntax: INSERT INTO table ["("list-of-columns")"] VALUES "(" list-of-expression ")"
Example:
INSERT INTO Student VALUES (12345678, 'Smith')
INSERT INTO Student (name, sid) VALUES ('Smith', 12345678)

Updating of tuples in a table / relation
Syntax: UPDATE table SET column"="expression {","column"="expression} [WHERE search_condition]
Example: UPDATE Student
SET address = '4711 Water Street' WHERE sid = 123456789

Deleting of tuples from a table / relation
Syntax: DELETE FROM table [WHERE search_condition]
Example: DELETE FROM Student WHERE name = 'Smith'

Primary and Foreign key SQL Example



DROP TABLE IF EXISTS tableName CASCADE;

```
CREATE TABLE Student ( sid INTEGER, … ,
   CONSTRAINT Student_PK PRIMARY KEY (sid)
);
CREATE TABLE UoS  ( ucode CHAR(8), … ,
   CONSTRAINT UoS_PK PRIMARY KEY (ucode)
);
CREATE TABLE Enrolled ( sid INTEGER, ucode CHAR(8), semester VARCHAR,
   CONSTRAINT Enrolled_FK1 FOREIGN KEY (sid) REFERENCES Student,
   CONSTRAINT Enrolled_FK2 FOREIGN KEY (ucode) REFERENCES UoS,
   CONSTRAINT Enrolled_PK  PRIMARY KEY (sid,ucode)
);
```